MASTER EN SCIENCES INFORMATIQUES

Distribution du système IDML - Interactive Data Manipulation Language - sur un réseau de micro-ordinateurs

Scoyer, Béatrice

Award date:
1982

Awarding institution:
Universite de Namur

FACULTES UNIVERSITAIRES NOTRE-DAME DE LA PAIX - NAMUR

INSTITUT D'INFORMATIQUE

DISTRIBUTION DU SYSTEME

IDML

- Interactive Data Manipulation Language -

SUR UN RESEAU DE

MICRO-ORDINATEURS

Mémoire présenté par

Béatrice SCOYER

en vue de l'obtention du titre de
Licencié et Maître en Informatique

Promoteur : J.L. HAINAUT

- Année académique 1981-1982 -

Je remercie toutes les personnes qui, à titre divers, ont participé à ce travail.

Je tiens en particulier à exprimer ma reconnaissance à Monsieur J.L. HAINAUT, promoteur de ce mémoire, pour sa disponibilité et ses conseils qui ont permis la réalisation de ce projet.

Je tiens à témoigner ma gratitude à Monsieur J.D. NICOUD pour m'avoir accueillie pour un stage à l'Ecole Polytechnique Fédérale de Lausanne.

Je remercie vivement l'ensemble des membres du Laboratoire de Micro-Informatique pour leur acceuil au sein de leur département.

Je tiens enfin à remercier Monsieur J.P. ADANS pour sa constante disponibilité, ainsi que ma famille pour le soin et la diligence apportés à l'édition de ce mémoire.

P L A N G E N E R A L

PREMIERE PARTIE

LE CONTEXTE REPARTI

SECONDE PARTIE

LE SYSTEME IDML MONO-UTILISATEUR

TROISIEME PARTIE

LE SYSTEME IDML PARTAGE.

QUATRIEME PARTIE

LE SYSTEME IDML DISTRIBUE

CINQUIEME PARTIE

EVALUATION DU SYSTEME IDML PARTAGE

INTRODUCTION.

Un survol de la littérature informatique des dernières années fait apparaître très clairement l'importance prise par les Bases de Données Réparties dans le domaine informatique.

Si ce phénomène est récent, il est pourtant incontestable que la première Base de Données Répartie est le monde réel: dès que l'on va chercher ailleurs une information qui manque, le concept Base de Données Répartie est mis en oeuvre.

Nous assistons donc à une nouveauté au plan informatique grâce à l'apparition d'outils adéquats. Cet atout technique est le développement des réseaux.

En effet, le réseau a permis à des ordinateurs de communiquer entre eux pour échanger des informations. Une fois que ces échanges se sont faits avec fiabilité, il est tentant de considérer l'ensemble des ordinateurs comme une seule installation informatique et d'y appliquer les concepts de Base de Données et Système de Gestion de Bases de Données : voici lancées les Bases de Données Réparties et les Systèmes de Gestion correspondants.

Si nous pouvons souligner des avantages des Bases de Données Réparties tant au plan humain qu'au plan économique ou organisation du système d'information dans les Entreprises, il faut encore en prouver la faisabilité.

La conjonction de deux éléments permet de penser que cette entreprise a des chances de succès. D'une part nous avons un acquis important sur les Bases de Données centralisées; cette connaissance doit évoluer et non pas être révolutionnée; d'autre part, les réseaux atteignent un bon degré de fiabilité.

Il n'en reste pas moins que le domaine de la recherche sur les Bases de Données Réparties est très vaste.

Ce travail se situe dans un cadre restreint de ce domaine.

Le point de départ a été la disponibilité d'un système portable de manipulation de Bases de Données hétérogènes (repris sous le nom de Système IDML (Interactive Data Manipulation Language)), spécifié par MM. J.L. HAINAUT et Y. DELVAUX à l'Institut d'Informatique (F.N.D.P. Namur).

Il présente une interface d'exploitation de Bases de Données existantes, gérées par des SGBD (Système de Gestion de Base de Données) hétérogènes. L'interface repose sur l'existence d'un modèle de données commun aux SGBD. Un langage de manipulation de données permet, tant au programmeur qu'à l'utilisateur non spécialiste, de travailler de manière interactive sur une ou plusieurs Bases de Données. Le système réalisant l'interface est indépendant des SGBD et de l'ordinateur sur lequel il fonctionne; il est en outre aisément extensible. Ce système a été conçu initialement pour un environnement mono-utilisateur.

Ce projet a pour objectif d'étudier pratiquement l'extension de ce système mono-utilisateur à un système distribué, capable de supporter une Base de Données Répartie.

La réalisation pratique a été faite sur un réseau local de micros-ordinateurs North-Star Horizon. Le logiciel de base du réseau a été réalisé par Mr. E. LEMAL dans le cadre du mémoire de fin d'études (Institut d'Informatique - F.N.D.P. Namur - juin 1982).
Le langage de programmation retenu est le Pascal MT+.

PRESENTATION DU TRAVAIL.

Dans la première partie, nous nous sommes penchée sur la littérature, afin de dégager les concepts principaux et les problèmes généraux en terme de Base de Données Répartie.

Ensuite, dans une seconde partie, nous présentons le Système de Manipulation de Bases de Données hétérogènes dans le contexte mono-utilisateur. Les spécifications ont été rassemblées à partir de diverses sources et ont fait l'objet d'une implémentation.

Une fois ces fondements établis, nous avons étendu ce système mono-utilisateur à un système partagé. Cette étude et son implémentation constitue la troisième partie.

Restait alors à continuer l'extension vers un système distribué. Des voies de recherche sont exposées dans la quatrième partie.

Enfin, avant de conclure le projet, nous avons évalué le système partagé dans trois dimensions : l'utilisateur, le réseau et le site de traitement.

P A R T I E 1. LE CONTEXTE REPARTI.

1.1. LES OBJECTIFS D'UNE BASE DE DONNEES REPARTIE.

Introduction

Les Bases de Données Réparties sont nées de la conjonction de deux phénomènes dans le domaine de l'informatique.

- d'une part l'importance accordée aux Bases de Données;
- d'autre part l'intérêt croissant pour la répartition.

Ce chapitre a dès lors pour but de rappeler succinctement quels sont les rôles attribués à une Base de Données (paragraphe 1.1.1.), de cerner le concept de distribution (paragraphe 1.1.2), pour en arriver à une définition globale d'une Base de Données Répartie (paragraphe 1.1.3.)

1.1.1. Les rôles d'une Base de Données et de son système de gestion.

Les objectifs assignés à la création d'une Base de Données dans une entreprise ou une organisation, sont de deux types:

1. représentativité d'un certain "monde réel" sur lequel fonctionnera une organisation,

2. optimisation de la fonction de disponibilité des données dans un système informatique.

Les données, les informations ne doivent donc pas être la propriété de quelques personnes dans une entreprise, mais mises à la disposition de plusieurs utilisateurs.

Le fait de partager les données est le reflet d'une certaine organisation de l'entreprise : on a constaté que la création de nouvelles applications engendrait ses propres fichiers. La création d'une Base de Données va à l'encontre de cette façon de faire puisqu'elle rend possible la centralisation, la coordination, l'intégration et la diffusion de l'information.

Les rôles d'une Base de Données et de son système de gestion sont dès lors multiples.

1. résoudre un problème de centralisation

   - de gestion des données : une équipe est responsable de l'administration de l'ensemble des données;
   - de la connaissance des données disponibles;
   - éviter le redondance.

2. résoudre un problème de sécurité

- prendre en compte la gestion de l'intégrité des données
  - cohérence des données quand l'information est partagée;
  - protection de l'information contre un mauvais fonctionnement du système.
- protéger les données contre des utilisateurs indiscrets.

3. résoudre un problème de disponibilité, de communication

il faut réduire la distance qui sépare les données de l'entreprise des utilisateurs de ces données.
- les utilisateurs sont diversifiés : ils ont des besoins et des qualifications distinctes;
- aux différents besoins doivent correspondre différents outils ou types de langage nécessaires pour communiquer.

4. résoudre un problème d'adaptation

le système doit s'adapter aux modifications de la structure même de la base de données. Ces modifications sont de deux types :
- changement de technologie;
- changement des besoins de l'utilisateur.

La notion de "data independance" est acquise en décomposant les systèmes de Base de Données en parties disjointes communiquant entre elles par des interfaces, permettant ainsi, lorsqu'une modification survient, de ne modifier que la partie en cause.

1.1.2. Le concept de distribution.

L'exigence de disponibilité informationnelle et les limitations technologiques (tant hard- que software) ont amené à une informatique centralisée, où une Base de Données est vue et implémentée comme un dépositaire central de l'information.

Les utilisateurs devenant de plus en plus conscients, et dès lors dépendants, de l'apport d'un tel système d'information, leurs exigences se sont accrues.

Les problèmes de temps de réponse, de disponibilité, de coût de l'information deviennent aigus.
Les applications se multiplient, le volume des données s'amplifie, les objectfis s'élargissent, un système centralisé ne suffit plus.

Le concept de distribution est une solution. Il peut prendre différents aspects :

- distribution des fonctions, pour décharger le système central (distribution hardware)

- distribution des systèmes, distribution des traitements

- distribution des données.

Ce dernier type de distribution est celui qui nous intéresse présentement. Il requiert un système d'information complet pour accéder aux données, là où elles résident.

Cette situation est la plus facile pour le programme, mais la plus complexe pour le système de gestion.

La disponibilité de réseaux d'ordinateurs a contribué à l'intérêt croissant porté aux Systèmes de Bases de Données Réparties.

L'élément du système qui s'occupe de la gestion locale de la Base de Données peut être :

- un périphérique spécialisé, sous le contrôle d'un processeur central
- un processeur à part entière, totalement dédié à la gestion des données.

1.1.3. Une Base de Données Répartie.

> Une Base de Données Répartie (BDR) est un ensemble de fichiers qui constituent une collection logique unique de données qui sont partitionnées, gérées et stockées en unités autonomes dans différents noeuds d'un réseau d'information.

Cette définition couvre tous les cas possibles, de la simple duplication de fichiers pour des raisons de sécurité ou pour une optimisation du trafic de communication, au cas le plus complexe d'éclatement de fichiers, qui peut être envisagé sur base du concept de "manipulation quasi locale" des données ou suite à la nécessité de les partager dans différentes occasions.

Une Base de Données globale peut exister dans deux types de systèmes distribués : hiérarchique ou horizontal.

Comparons-les succinctement à un système centralisé.

1. Architecture centralisée.

   Tout le travail est réalisé à un centre de traitement. Les utilisateurs éloignés sont servis via des liens de communication entre eux et le centre.

   Les avantages :

   - économie (en coût d'exploitation, voire même coût hardware);
   - justification économique d'un software "élégant";
   - facilité relative pour maintenir une Base de Données cohérente.

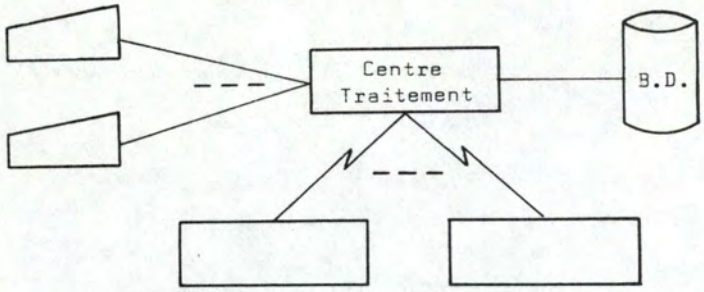


Fig. I.1. - Architecture centralisée.

Les inconvénients :

- vulnérabilité vis-à-vis des défaillances
- incapacité à répondre rapidement à un changement technologique ou utilisateur;
- la complexité de l'exploitation s'accroît avec le nombre d'utilisateurs;
- le coût de transfert des données.

2. Architecture distribuée hiérarchiquement

Les processeurs partagent les tâches de façon structurée. Chaque composant est contrôlé par les membres du niveau suivant dans la hiérarchie. (voir Fig. I.2.)

Les avantages :

- robustesse;
- diminution des coûts de communication

L'inconvénient :

- difficulté d'employer des composants substituts, surtout dans les niveaux supérieurs

3. Architecture distribuée horizontalement

Tous les processeurs coopèrent à un niveau égal, pour réaliser un ensemble de tâches. Aucun composant n'exerce un contrôle sur les autres composants. (voir Fig. I.3.)

Les avantages :

- indépendance d'un processeur;
- possibilité d'employer de multiples "fournisseurs";
- réponse rapide à un changement;
- accroissement de la participation et de la responsabilité locale.

Les inconvènients :

- personne, aucun "fournisseur" ne supporte le système dans sa totalitè;
- difficultè de convertir un système existant (comparè au système distribuè hièrarchiquement).

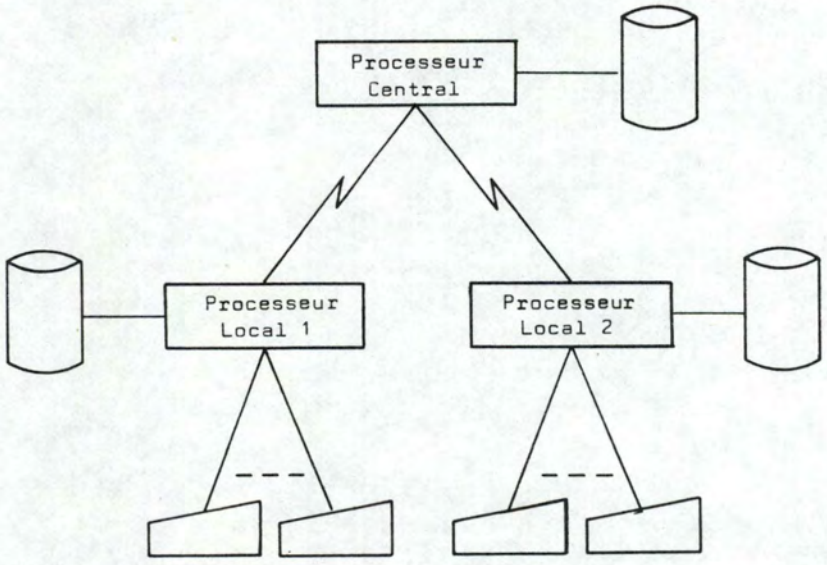


Fig. I.2. - Architecture distribuée hièrarchiquement. distribution verticale à deux niveaux.

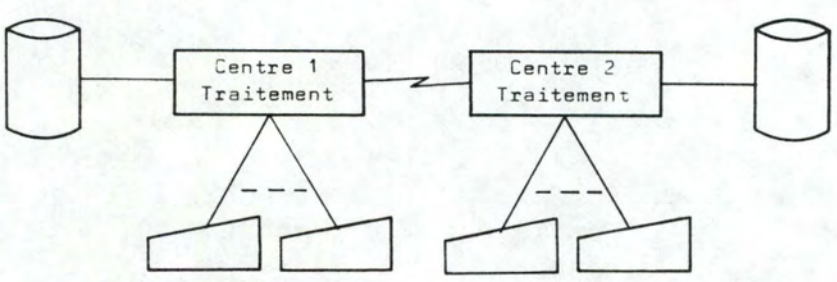


Fig I.3. - Architecture distribuée horizontalement.

1.2. IMPLEMENTATION D'UN SYSTEME DE GESTION DE BASE DE DONNEES REPARTIE.

Introduction.

La conception d'un SGBDR pose de nombreux problèmes qui font l'objet de multiples études et pour lesquels il n'existe pas de solution qui reçoivent une approbation unanime.

Nous nous intéresserons à cinq aspects particuliers :

- l'architecture générale du système doit-elle tenir compte ou non de systèmes locaux préexistants ?
  et quels sont les problèmes qui y sont liés (paragraphe 1.2.1.)

- comment contrôler les données ?
  les problèmes posés par l'allocation de copies multiples et l'éclatement des fichiers (paragraphe 1.2.2.)

- comment retrouver et manipuler les données ? (paragraphe 1.2.3.)

- l'interaction homme-machine et machine-machine :
  le langage de communication (paragraphe 1.2.4.)

- la composante réseau: le protocole de communication (paragraphe 1.2.5.)

1.2.1. Système réparti et systèmes locaux.

Doit-on tenir compte de l'existence de systèmes locaux lors de la conception du système réparti ?

La solution à cette première question dépend souvent des dimensions des machines impliquées dans le système et de facteurs politiques :

Si les machines sont des ordinateurs importants, fournis par les constructeurs avec un software standard, on pourait envisager de modifier le SGBD existant et/ou le système d'exploitation, pour les ajuster à l'environnement distribué.

Mais il paraît préférable de construire un logiciel, tournant comme un processus utilisateur permanent, pour gérer les communications de haut niveau entre les ordinateurs et servir d'interface entre différents SGBD. Le composant SGBD doit cependant être assez compatible au niveau des fonctions qu'il réalise avec les autres SGBD.

SGBD : Système de Gestion de Base de Données,
SGBDR : Système de Gestion de Base de Données Répartie.

D'autre part, la tendance actuelle est à la dispersion de la puissance de l'ordinateur. Des réseaux de mini-ordinateurs se substituent à des machines de taille importante. Dans ce cas, on pourrait envisager de dédier un mini-ordinateur pour ces fonctions particulières et dès lors construire un software spécialisé pour un SGBDR.

La solution que nous retiendrons consiste à tenir compte des systèmes locaux existants. Le passage à un environnement réparti ne doit pas être une révolution pour l'organisation.

Si l'indépendance physique/logique préserve les programmes d'application vis-à-vis des changements dans la structure de stockage et des changements dans le modèle de définition de données, et ce dans un contexte "stand alone", cette indépendance reste un objectif principal dans un contexte réparti.
En outre, il apparaît absurde de négliger tout ce qui existe afin de passer à une normalisation qui serait abusive.

Dès lors, un système de gestion de Base de Données Réparties devra tenir compte de deux éléments fondamentaux :

- les SGBD locaux sont hétérogènes;
- les machines impliquées sont différentes, avec leurs caractéristiques propres,

ce qui implique des problèmes de traduction et de conversion.

En ce qui concerne la coopération entre SGBD hétérogènes, nous envisagerons un langage de manipulation de données, interface entre l'utilisateur et les SGBD concernés, quels qu'ils soient.

1.2.2. Le contrôle des données

De nombreux algorithmes ont été élaborés afin de proposer une solution au problème de l'allocation des fichiers dans un réseau d'ordinateurs.

Ils prennent en compte des critères, tels que minimisation du stockage et des coûts de transmission, minimisation du temps de réponse.

En fait, l'existence de plus d'une copie d'un fichier dans le réseau, même si elle est justifiée d'un point de vue économique, peut causer de nombreux troubles:

- incohérence dans l'espace : toutes les copies ne sont pas identiques;
- incohérence dans le temps : ordre des requêtes;
- accès concurrents parallèles

Le contrôle des données prend avec ces problèmes de copies multiples toute son importance.

Un autre point qui nécessite la présence de la fonction de contrôle, est celui de l'éclatement d'un fichier sur différents sites.

Comment s'assurer qu'une opération effectuée sur un site local est totalement cohérente vis-à-vis du reste de la Base de Données Répartie ?

Le contrôle d'un SGBDR peut être centralisé ou réparti :

centralisé : un seul centre joue le rôle de l'administrateur de la Base de Données Répartie. C'est par lui seul qu'est défini et modifié le schéma. Il gère les copies multiples et veille à la cohérence globale de la Base de Données Répartie.

réparti : tous les sites gardent le contrôle sur l'ensemble de leurs données. Mais comme ils sont destinés à réaliser un service intégré, il est indispensable qu'ils communiquent entre eux, afin que le système reste cohérent.

1.2.3. Localisation des données

Comment retrouver une information sur le réseau ?

La première solution consiste à avoir sur chaque site une copie du catalogue du réseau. Ce catalogue peut être divisé en deux niveaux :

- le premier contenant des informations détaillées sur les données locales;
- le second contenant des renseignements sur la localisation des données externes.

La solution opposée est de ne pas avoir de catalogue du tout. Dans ce cas, chaque site garde seulement le répertoire de ses données locales. Un site local qui requiert un accès à des données externes diffuse sa demande à tous les noeuds du réseau et il attend une réponse positive. Si la donnée est stockée sur plusieurs sites, il faut sélectionner dynamiquement un ou plusieurs noeuds et traiter normalement avec eux.

Une solution intermédiaire est d'avoir un catalogue central sur un site particulier connu de tous les autres noeuds.
Deux solutions sont encore envisageables :

- ce site particulier n'aura qu'un rôle d'évaluateur : pour une donnée, il fournira au site demandeur la (les) localisation(s)
- ce site particulier sera à la fois évaluateur et distributeur : il répercutera la requête du site demandeur au(x) site(s) concerné(s).
  A noter que dans ce dernier cas, ce site sera également responsable du contrôle des données.

1.2.4. L'interaction homme-machine : le langage de communication.

La manipulation de données par des utilisateurs implique l'existence d'outils d'interaction avec la Base de Données.

Les opérations disponibles à l'utilisateur sont la mise à jour (création, modification, suppression) et la recherche d'un objet.

Mais l'utilisateur a une vue abstraite de la Base de Données. Il lui est donc impossible d'agir directement sur un objet puisqu'il n'en connaît pas la structure détaillée.

Pour pallier cet obstacle, il est nécessaire d'avoir des opérations qui s'adaptent à l'image de l'objet : ce sont les opérateurs prè-définis de haut-niveau regroupés dans le Langage de Manipulation de Données.

Ce langage peut être de deux types :

- il se compose d'opérations sur la Base de Données, qui doivent être incluses dans un langage standard de programmation de haut niveau.
  Il s'agit, par exemple, du Langage de Manipulation de Données offert par le SGBD CODASYL, qui s'intègre dans la programmation en COBOL;

- il constitue un langage à part entière, qui permet l'accès aux données d'une façon aussi simple et naturelle que possible et qui offre simultanément des mécanismes de programmation puissants.
  Nous trouvons dans cette classe des langages tels que SEQUEL (Structured English QUEry Language) et IDML (Interactive Data Manipulation Language).

Pour pouvoir être exploité par la machine, ce langage de requête doit être transformé en un langage intermédiaire (appelé "code interne" ou "langage de base"), tel que le langage machine.

Ce langage intermédiaire comprend des instructions arithmétiques, des instructions de tests et de branchements, des directives de définitions de constantes et de zones mémoires de travail, et des appels à des fonctions externes particulières sur les Bases de Données : ce sont les primitives de base offertes par le SGBD.

Dès lors, nous pouvons envisager que l'utilisateur, suivant sa connaissance du système travaille avec le système à trois degrés de dialogue.

De même, dans le contexte réparti, la demande de travail pourra se situer à un des trois niveaux suivants, chaque niveau ayant des caractéristiques propres (performance au plan trafic, temps exécution, charge du site...)

1. L'utilisateur s'exprime par primitive

   Cette approche pas-à-pas exige une interaction constante entre la machine et l'utilisateur, ce dernier devant réagir dynamiquement à chaque réponse de la machine.

2. L'utilisateur s'exprime dans le langage intermédiaire

Il s'agit pour lui d'établir un algorithme de recherche :
- sélection des articles à traiter,
- traitement de ces articles,
- production d'un rapport.

Cette méthode nécessite des connaissances en technique de programmation et surtout une connaissance précise du système.

3. L'utilisateur s'exprime dans le langage de haut niveau.

Il spécifie les critères de recherche plutôt que la procédure de recherche de l'information.
Le travail de la machine est dès lors considérable, (décomposer une requête, distribution du traitement, organisation des réponses), la gestion de la Base de Données étant totalement transparente à l'utilisateur.

1.2.5. Le support réseau : comment communiquer.

Une Base de Données Répartie nécessite un support de communication : le réseau.
Et pour que les systèmes puissent communiquer, il a fallu établir des règles.

L'Organisation Internationale de Normalisation s'est intéressée à l'interconnexion de systèmes ouverts (ISO).
L'objectif d'ISO est d'obtenir des échanges d'informations fiables entre n'importe quel processus d'application appartenant à n'importe quel système.

Pour ce faire, il a établi un modèle de référence. Ce modèle consiste en une description logique et fonctionnelle des fonctions d'échange et en un essai de standardisation dans la conception des protocoles.

L'architecture adoptée par ISO fait intervenir le concept de couches.
Chaque couche réalise un ensemble bien défini de fonctions, en utilisant un autre ensemble bien défini de fonctions fournies par la couche inférieure.
Ces fonctions forment un ensemble de services qui peuvent être demandés par la couche supérieure.

Cette architecture standard est définie en sept couches :

1. La couche application comprend les programmes d'application avec leurs conventions d'échanges et de coopération.
   On y trouvera donc un protocole spécifique selon le type d'application.

2. La couche présentation est responsable de la présentation des données échangées par les applications de manière à résoudre les différences syntaxiques ou sémantiques, tout en gardant le sens de l'information.

3. La couche session fournit aux couches de présentation qui coopèrent, les moyens d'organiser et synchroniser le dialogue, et de contrôler les échanges de données.

4. La couche transport est responsable du contrôle du transport d'information de bout en bout, entre deux processus, au travers d'un réseau. Les utilisateurs sont connus par une adresse.

5. La couche réseau s'occupe des problèmes de routage et d'aiguillage associés à une connexion réseau.

6. La couche liaison de données est responsable de l'acheminement de blocs d'informations sur des liaisons de données.

7. La couche physique comprend les moyens nécessaires à la transformation de bits d'information.

Deux types de réseaux supportant une Base de Données Répartie peuvent être envisagés :

- un réseau général reprenant toutes les couches décrites ci-avant;

- un réseau à portée locale défini sur une architecture simplifiée:

  - en premier lieu, nous retrouverons la couche application qui comprend de manière générale les processus d'application qui communiquent. Ces processus réalisent eux-mêmes les fonctions de session et/ou de présentation, selon nécessité.
  - viendront ensuite les couches transport, liaison de données et physique qui fournissent un service aux processus d'application.

1.3. GESTION D'UNE BASE DE DONNEES.

Introduction

Les problèmes présentés lors du chapitre précédent l'ont été en dehors de toute considération d'architecture.

Rappelons maintenant les différents niveaux de cette architecture, tels qu'ils ont été proposés par le groupe d'étude ANSI/X3/SPARC (paragraphe 1.3.1.)

Comparons ensuite cette proposition avec ce qu'offrent actuellement les SGBD (paragraphe 1.3.2.)

Nous terminerons ce chapitre en énonçant l'évolution possible de cette architecture dans le cadre d'une Base de Données Répartie (paragraphe 1.3.3.)

1.3.1. Proposition théorique d' ANSI/SPARC

Trois niveaux de conception ont été mis en évidence :

1. une description abstraite, élaborée dans les termes d'un modèle, de la structure logique de la Base de Données dans sa totalité. Ce modèle conceptuel est donc la description de l'organisation aussi stable qu'elle, les données y sont claires et précises. Elles sont une explication du réel perçu.
   Le texte qui décrit ce modèle est le Schéma Conceptuel (SC).

2. une description de la structure du stockage physique des données composant la Base de Données.
   Ce Schéma Interne (SI) qui est une description technologique de la Base de Données est dérivé de la structure conceptuelle.

3. une description de la structure logique de la Base de Données ou d'une partie de celle-ci, telle qu'elle est vue par un utilisateur particulier.
   On dégage ainsi un Schéma Externe (SE) approprié à chaque utilisateur.

Dans une telle découpe, l'indépendance entre l'utilisateur et l'implémentation est assurée : chaque niveau est indépendant des autres niveaux, les liens sont établis par mapping.

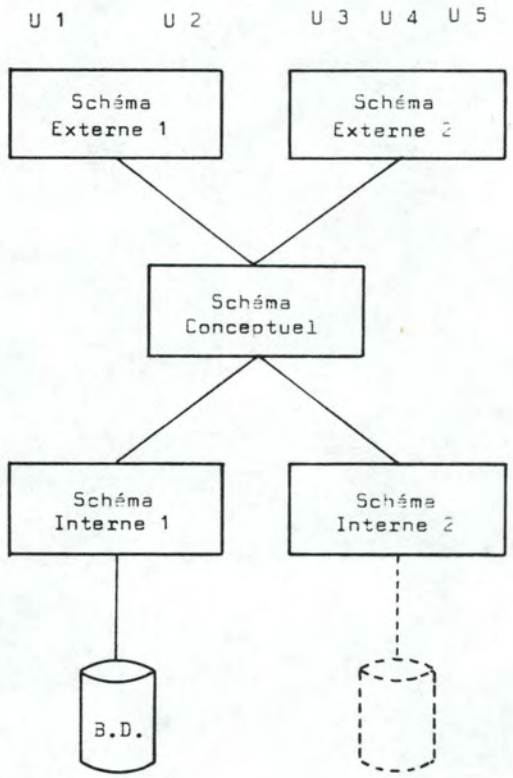


Fig. I.4. - Les trois niveaux de conception. proposition ANSI/SPARC

1.3.2. Perception actuelle.

Mais dans la réalité, le schéma conceptuel n'est pas encore implémenté. Les Systèmes de Gestion de Bases de Données offrent actuellement un Schéma des Accès Logiques.

Le rôle de la structure d'accès est de décrire les chemins d'accès aux données. Les seuls accès auxquels pourra faire appel le programmeur sont ceux décrits dans cette structure.
Tous les résultats attendus doivent pouvoir être extraits de la structure d'accès.

Cette structure est formée d'un ensemble de relations et décrit la façon dont les contraintes d'intégrité vont être vérifiées.

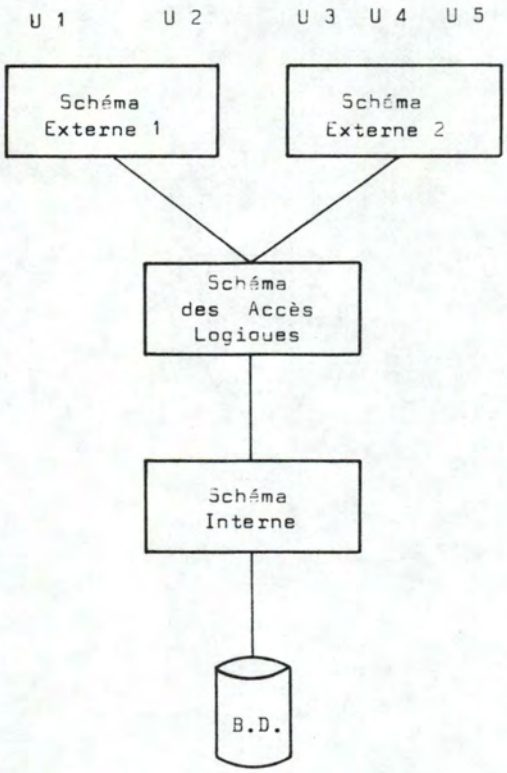


Fig. I.5. - Les trois niveaux de conception.
perception actuelle.

1.3.3. Schèmas d'une Base de Donnèes Rèpartie.

Pour faire coopèrer des systèmes locaux, il faut qu'ils partagent une description commune de la Base de Donnèes Rèpartie : le schèma global de la Base de Donnèes Rèpartie, ou Schèma Conceptuel Rèseau. Il dècrit l'ensemble des donnèes dans une vue globale, sans souci du (des) noeud(s) dans le(s)quel(s) elles sont stockèes.

Quant à la structure d'organisation du stockage physique des donnèes, elle existe au sein des systèmes locaux (schèma interne de chaque site). Le Schèma Interne Rèseau contiendra les informations relatives à la localisation des donnèes (le ou les lieux de stockage, l'èclatement des fichiers,...)

Enfin, par rèfèrence au schèma conceptuel rèseau, on dèfinira les Schèmas Externes de la Base de Donnèes Rèpartie, dècrivant celle-ci dans la vue des applications qui l'utilisent.

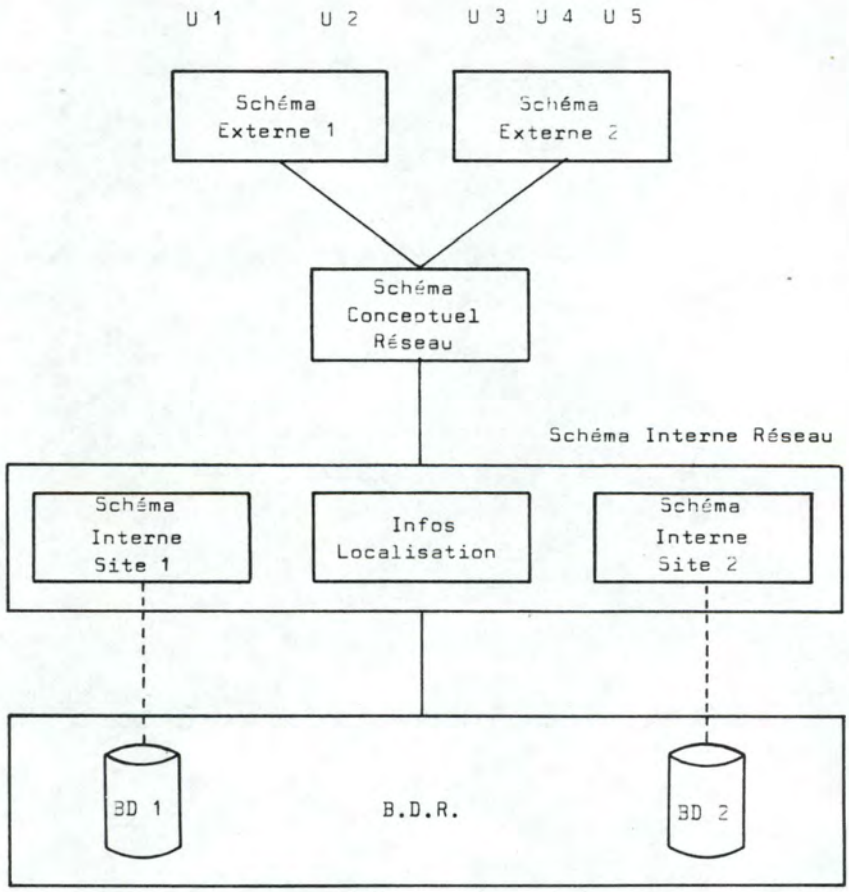


Fig. I.6. - Les trois niveaux de conception.
proposition pour une BDR.

1.4. UNE IMPLEMENTATION PRATIQUE : LE PROJET PHLOX.

Introduction.

Avant d'entrer pleinement dans le projet qui nous occupe, il nous a paru intéressant de terminer cette introduction au contexte Base de Données Répartie, par la présentation du projet PHLOX, réalisé par B. DEL VECHHIO et P. PENNY à l' I.N.R.I.A. (France).

Ce projet a conduit à l'élaboration de trois logiciels de Gestion de Base de Données pour micro-ordinateurs.

- PHLOX1 pour une machine individuelle (BD mono-utilisateur)
- PHLOX2 pour un serveur Base de Données sur un réseau local (Base de Données partagées)
- PHLOX3 pour une station de travail interconnectée à d'autres, grâce à un réseau (Base de Données Répartie).

Si nous avons choisi de vous décrire cette réalisation en particulier, c'est parce qu'elle présente quelques éléments semblables à ceux que nous retrouverons dans le cadre du projet de distribution du système IDML :

- une implémentation sur micro-ordinateur;
- l'utilisation d'un réseau local;
- la construction par étapes d'un système distribué (d'abord un système mono-utilisateur, puis un système partagé et enfin un système distribué);
- un objectif de facilité d'utilisation pour l'utilisateur inexpérimenté, tout en offrant un système puissant.

Le projet PHLOX a été développé à l'Institut National de Recherche en Informatique et Automatique (INRIA-France) dans le cadre des projets pilotes SIRIUS (Base de Données Distribuées) et KAYAK (Bureautique).

1.4.1. Architecture du système PHLOX.

La découpe en niveaux et la terminologie correspondent aux propositions du groupe ANSI/SPARC.

1. le niveau conceptuel, qui utilise le modèle relationnel;
2. le niveau des accès logiques internes, qui se base sur des extensions du modèle des SGBD en réseau;
3. le niveau physique interne.

La Base de Données est stockée suivant le schéma en réseau, mais une interface relationnelle et une interface en réseau sont disponibles pour l'utilisateur.

Le Langage de Description des Données (LDD) est basé sur trois concepts :

- le type de données,
- le type d'article,
- le type de relation binaire.

Un concept sous-jacent est le set de réalisations de type d'article.

Un aspect important du système est que la description de la Base de Données peut être modifiée à chaque niveau avec un minimum de perturbation.

1.4.2. Langages de Manipulation des Bases de Données.

Le Langage de Manipulation de Données relationnel offre un ensemble de primitives qui opèrent sur la Base de Données en réseau. Un modèle de mapping transforme la description logique relationnelle en une description logique réseau.

Le Langage de Manipulation de Données navigationnel comprend toutes les commandes nécessaires pour parcourir la Base de Données. Il comporte en outre différentes variables utilisées pour mémoriser plusieurs réalisations courantes de types d'articles.

Modes d'utilisation :

Les commandes du Langage de Manipulation de données peuvent être directement interprétées à partir d'une console ou d'un fichier de commandes. Des commandes spéciales du système permettent d'éditer les résultats des requêtes. Le Langage de Manipulation de Données navigationnel peut être interprété mais le Langage de Manipulation de Données relationnel est plus approprié pour ce travail interactif.

Une application peut utiliser le Langage de Manipulation de Données dans un langage de programmation tel Pascal ou PL1. Les commandes navigationnelles sont ici très efficaces.

1.4.3. Cohérence de la Base de Données.

Le système PHLOX inclut un mécanisme de "contexte virtuel" pour assurer la cohérence de la Base de Données. Les modifications apportées par un utilisateur sont gardées séparées de la Base de Données elle-même, jusqu'à ce que les contrôles soient effectués.

1.4.4. Le logiciel PHLOX1 - mono-utilisateur

Le système PHLOX1 a trois fonctions.

1. un interface réservé à l'Administrateur de la Base de Données qui travaille comme un interpréteur de commande du Langage de Description des Données, ou sur un mode de question/réponse;

2. un interpréteur des commandes du Langage de Manipulation de données pour l'utilisateur;

3. une unité d'exécution de programmes d'application précompilés.

Le système ne travaille que pour un utilisateur à la fois, mais peut gérer plusieurs Bases de Données disjointes.

1.4.5. Le logiciel PHLOX2 - système partagé.

PHLOX2 est construit pour un micro-ordinateur utilisé comme un serveur de Base de Données partagée par plusieurs programmes d'application qui s'exécutent sur d'autres micros-ordinateurs connectés à un réseau.

Les programmes d'application sont appelés producteurs de transactions et PHLOX2 en est le consommateur.

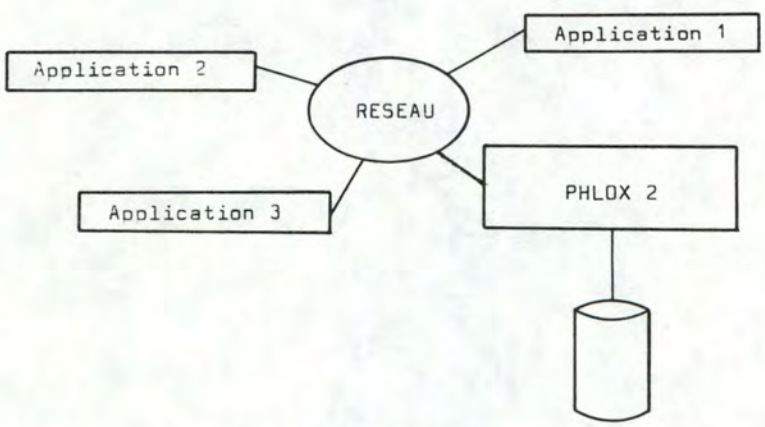


Fig. I.7. - PHLOX2 : système partagé.

En plus des fonctions présentes dans PHLOX1, le logiciel PHLOX2 offre deux fonctions supplémentaires :

1. un protocole de communication réseau;

2. un protocole de communication appelé "système d'exploitation distribué":
   - il identifie les transactions générées;
   - il active, contrôle et détruit des tâches pour exécuter les transactions;
   - il encode et décode les appels des programmes d'application;
   - il rend transparent pour chaque programme d'application les reprises en cas d'incident.

1.4.6. Le logiciel PHLOX3 - système distribué.

Les stations de travail sont des micros-ordinateurs interconnectés via un réseau.

PHLOX3 permet la description de la Base de Données Distribuée avec un schéma global et la localisation des données sur les sites locaux.

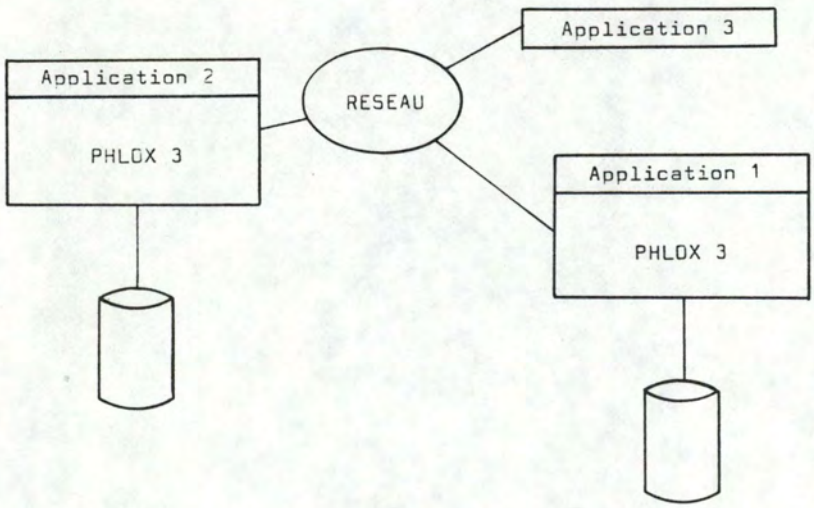


Fig. I.8. - PHLOX3 : système distribué.

Il réalise la scission des requêtes globales suivant la localisation des données; il demande l'exécution des parties de la requête sur chaque site concerné et rassemble les résultats en contrôlant la cohérence de la Base de Données Distribuée.

P A R T I E 2. LE SYSTEME IDML MONO-UTILISATEUR.

2.1. PRINCIPES ET ARCHITECTURE DU SYSTEME IDML.

Introduction.

Ce chapitre présente les principes et l'architecture d'un interface d'exploitation de Bases de Données existantes, gérées par des Systèmes de Gestion de Bases de Données hétérogènes.

Dans un premier temps, nous regarderons l'objectif poursuivi par ce projet (paragraphe 2.1.1.).
Ensuite, nous décrirons le modèle des données sur lequel repose l'interface (paragraphe 2.1.2.).
Enfin, nous introduirons les différents éléments qui constituent l'architecture du Système IDML (paragraphe 2.1.3.).

Cette partie a donc pour rôle de vous présenter le système IDML. Les spécifications qui définissent le système ont fait l'objet de différents documents. Cette partie a dès lors consisté à les rassembler et à en tirer les idées essentielles pour réaliser un des objectifs qui nous étaient fixés, à savoir l'implémentation de ce système sur un micro-ordinateur.

Voici ces divers documents dans lesquels vous pourrez trouver des informations plus complètes et les justifications rigoureuses de certaines options qui ont été retenues.

- Y. DELVAUX - JL. HAINAUT
  "Système portable de Manipulation de Bases de Données hétérogènes."
  FNDP - NAMUR Mars 1981

- JL. HAINAUT
  "Projet IDML. Implémentation des programmes IDML."
  FNDP - NAMUR Juin 1979

- Y. DELVAUX
  "Projet IDML. Le langage de base."
  FNDP - NAMUR Janvier 1980

- Y.DELVAUX
  "Projet IDML. La représentation des données."
  FNDP - NAMUR Janvier 1980

- JP. ROBERT
  "Génération automatique d'interfaces d'accès à des Base de Données COBOL" - Chapitre 2. Les interfaces.
  FNDP - NAMUR Septembre 1981

2.1.1. Objectif du projet IDML

Le projet IDML (Interactive Data Manipulation Language) a pour objectif la conception et la réalisation d'un système interactif d'exploitation de Bases de Données.

Avec pour contraintes ;

- l'accès aux Bases de Données doit être offert tant à l'usager occasionnel qu'au programmeur d'application;
- le système doit prendre en charge toute Base de Données existante et non pas exiger des Bases de Données qui lui sont adaptées;
- il doit permettre la coexistence de Bases de Données hétérogènes, c'est-à-dire gérées par des SGBD différents;
- le système doit être transportable vis-à-vis des machines et de SGBD;
- il doit être extensible.

La première tâche de conception a été la mise au point d'un modèle général des données qui offre à l'utilisateur une description des Bases de Données qui soit simple, indépendante des LDD (Langage de Description de Données), et suffisamment précise pour permettre une exploitation efficace de ces Bases de Données.

La deuxième tâche fut la définition du LMD (Langage de Manipulation de Données) qui permet l'accès aux données vues au travers du modèle général. Ce langage devait offrir des possibilités d'accès sous une forme aussi simple et naturelle que possible, tout en offrant des mécanismes de programmation puissants.

Enfin, l'architecture du système IDML a été élaborée de manière à mettre en oeuvre modèle et langage de manipulation tout en respectant les objectifs d'indépendance, de simplicité d'utilisation et d'extensibilité.

2.1.2. Le modèle de données : le modèle d'accès.

Il convient de préciser le niveau du modèle de données. Il serait tentant d'offrir d'une Base de Données opérationnelles une description conceptuelle qui pourrait s'exprimer dans l'un des modèles actuellement répandus : modèle de Codd, modèles Entité/Association, modèles binaires,...

Nous nous exposerions cependant à d'énormes difficultés de traduction des expressions de manipulation de données liées à ce modèle en séquences d'appels au SGBD.

Ces difficultés sont de deux ordres :

- les premières concernent l'efficacité de la traduction, et ce problème n'est pas encore réellement résolu;

- les secondes tiennent à l'objectif même du projet : prendre en charge des Bases de Données opérationnelles; la conceptualisation d'une telle base masquerait un nombre considérable de détails techniques dont la connaissance est cependant indispensable pour manipuler les données, particulièrement en ce qui concerne la mise à jour.

Il faut donc viser un autre niveau de perception : nous ferons appel au modèle des accès logique.

Les concepts essentiels sont organisés en trois classes : les objets, les mécanismes d'accès et les primitives.

1. Les objets.

L'élément de base est l'article, unité d'information enregistrée qui peut faire l'objet d'une demande d'accès, de création, de modification et de suppression. Tous les articles sont distincts et chacun appartient à un type d'article qui en définit les propriétés générales.

La valeur d'item est une donnée manipulable dans un programme et qui appartient à un type appelé item. Un item est décomposable en d'autres items si ses valeurs sont constituées de fragments significatifs. A un article peuvent être associées des valeurs d'item. Les propriétés d'association entre un type d'article et un item font de ce dernier un item obligatoire ou facultatif, simple ou répétitif.

Un fichier est une collection dynamique d'articles; un article appartient toujours à un fichier.

Une Base de Données est la collection des articles d'un ensemble de fichiers; elle contient toujours un article particulier ("article Système") qui est son point d'entrée et portera le nom de cette Base de Données, ou plus exactement de son schéma externe. Certains SGBD donnent effectivement une existence matérielle à cet article (IDS, DMS11, CODASYL).

Afin de simplifier la manipulation des fichiers pour l'utilisateur non spécialiste (cette notion ne peut lui être cachée) on associera à tout type d'article un item (fictif) de nom "fichier", dont la valeur est le nom du fichier de l'article. Ceci rend très homogène l'accès aux articles, tant en sélection qu'en mise à jour.

2. Les mécanismes d'accès.

L'accès est pour un programme la mise à disposition d'un objet de la Base de Données. La possibilité de cette opération est représentée par l'existence d'un mécanisme d'accès parmi les suivants :

- l'accès à une Base de Données, représenté par l'accès à son "article Système";
- l'accès à un fichier qui peut être simulé grâce à l'item "fichier";
- l'accès aux articles d'une Base de Données d'un fichier;
- l'accès aux valeurs d'items d'un article;
- l'accès aux articles auxquels est associée une valeur déterminée d'un item (qui est appelé clé d'accès de ce type d'article);
- l'accès à des articles à partir d'un article; ce mécanisme, appelé chemin d'accès, mérite une définition plus précise.

Un chemin d'accès est un mécanisme qui associe à un article (origine) 0, 1 ou plusieurs articles (cibles), d'une manière telle qu'il soit possible d'accéder à partir de l'article origine successivement aux cibles.

Tout chemin appartient à un type défini essentiellement par :
- les types possibles de l'article origine;
- les types possibles des articles cibles;
- sa connectivité (N-N, 1-N, N-1, 1-1, de l'origine vers la cible);
- l'existence d'un type de chemin inverse.

3. Les primitives

Elles correspondent aux opérations élémentaires qu'il est possible d'effectuer sur les objets d'une Base de Données. Elles peuvent être rangées en trois classes :

- les primitives d'accès qui mettent en oeuvre les mécanismes d'accès;
- les primitives de modification qui font évoluer l'état de la Base de Données;
- et les primitives de contrôle d'environnement, qui fonctionnellement permettent la création de super-primitives, l'établissement de points de reprise et la maîtrise de la concurrence (notion de transaction dans certains SGBD).

2.1.3. Le système IDML.

L'objectif du système IDML est le traitement des programmes IDML à leurs différents stades. Il doit, par conséquent, assurer le dialogue avec l'utilisateur, gérer l'introduction, la correction et la conservation des programmes, assurer et contrôler l'exécution et la mise au point de ceux-ci, et donc, gérer les communications avec les SGBD. Il doit également réaliser la traduction des schémas des SGBD en schémas simplifiés selon le modèle d'accès.
Le système devrait, dès le départ, être doté de caractéristiques d'indépendance ou portabilité vis-à-vis de la machine et des SGBD, d'extensibilité, et, enfin, de généralité et de simplicité de concepts.

Architecture du système IDML.

L'architecture repose sur les principes suivants :

- toute communication avec l'extérieur est perçue comme une transaction avec une Base de Données au sens du modèle d'accès. Tel est évidemment le cas des Bases de Données authentiques, mais aussi en est-il ainsi du terminal, des fichiers de travail, des sauvegardes des programmes sources et objets et de la base des schémas. Ces communications sont gérées par des interfaces d'accès, dont les points d'entrée correspondent aux primitives du modèle d'accès.

- toute opération de manipulation de données qui dépend de la machine et du système d'exploitation est réalisée par un module spécialisé. Outre les interfaces d'accès déjà mentionnées, on trouvera dans le système un module arithmétique, un module de manipulation de chaînes de caractères, etc...

- l'interprétation d'un programme IDML conduirait à des performances dégradées; par contre, il n'est pas question de réaliser une compilation en un langage classique, en raison du caractère interactif du système. L'exécution est, par conséquent, décomposée en deux phases : une traduction dans un code intermédiaire, puis une interprétation de celui-ci.

Le système IDML est constitué de deux sous-systèmes : le système de traitement des programmes et celui du traitement des schémas.

Le sous-système de traitement des programmes.

Les composants fondamentaux sont : un éditeur de textes, un compilateur, un relieur, un chargeur et un interpréteur.

L'éditeur, inspiré du BASIC, permet d'entrer des procédures IDML, de les corriger, de les stocker dans un fichier.

Le compilateur traduit une procédure IDML en une procédure en code interne (qui peut être stockée dans un fichier).

Cette traduction nécessite la connaissance du schéma des accès logiques des bases exploitées; celui-ci est accessible dans la Base de Données des schémas IDML. Dans le code interne, les variables sont référencées via des descripteurs qui précisent type, longueur, structure de décomposition et adresse absolue des valeurs. Ajouté à l'adressage relatif des branchements, ceci rend le code interne aisément relogeable.

Le relieur assemble, si nécessaire, les différentes procédures pour constituer un programme en code interne exécutable.

Le chargeur charge dans la mémoire de la machine virtuelle le programme en code interne.

L'interpréteur, ou machine virtuelle IDML, exécute le programme en code interne. Il fait appel à des processeurs spécialisés, tels que le module arithmétique, le module de manipulation de chaînes ainsi que le processeur de Bases de Données qui regroupe les différents interfaces d'accès aux Bases de Données, au terminal, aux différents types de fichiers.

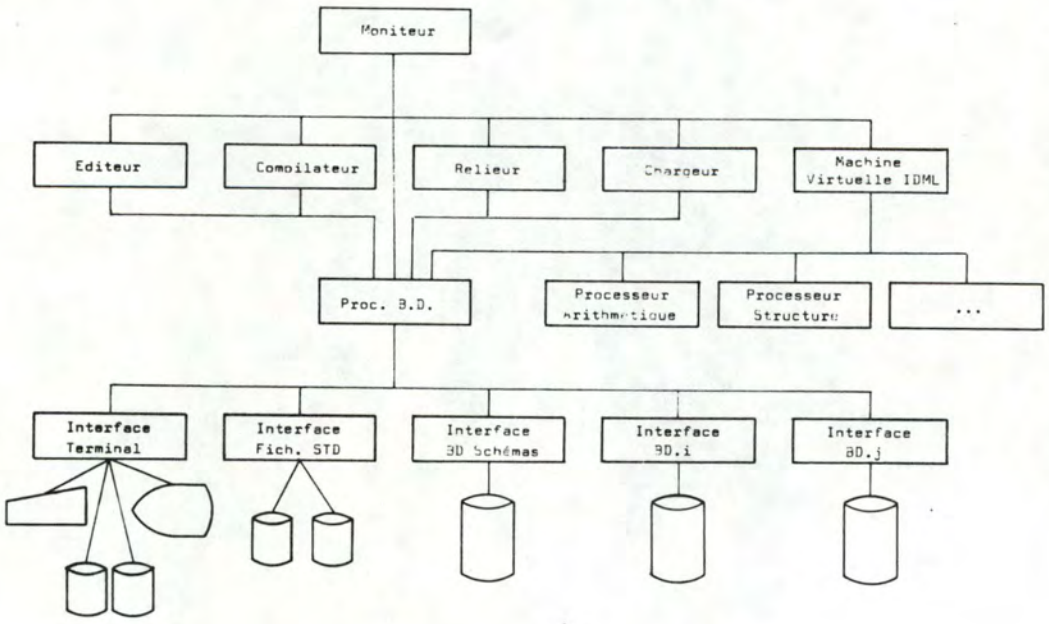


Fig. II.1. - Le sous-système de traitement des programmes.

Le sous-système de traitement de schémas.

La fonction de ce système est la production de deux composants du sous-système précédent, à savoir la Base de Données des schémas IDML et les interfaces d'accès.

Ces deux composants ont, en fait, une portée beaucoup plus générale que celle du système IDML. en effet, les interfaces offrent au programmeur d'application COBOL (ou autres) un très haut niveau d'indépendance par rapport au SGBD, tandis que la Base de Données des schémas pourra servir à la production de documents à destination de ces programmeurs ou même à la génération automatique de programmes d'exploitation de Bases de Données.

Ces deux composants sont produits par un programme qui, à partir du texte rédigé dans le LDD du SGBD, range dans la Base de Données des schémas la description du modèle d'accès sous-jacent à ce texte, puis construit l'interface d'accès à la Base de Données; cette construction, très systématique, consiste à associer à chaque primitive la séquence d'instructions COBOL + LMD du SGBD qui le réalise.

Ce programme est dédicacé à un SGBD. Il est constitué d'un analyseur, d'un chargeur (BD de schémas) et d'un générateur (d'interfaces).

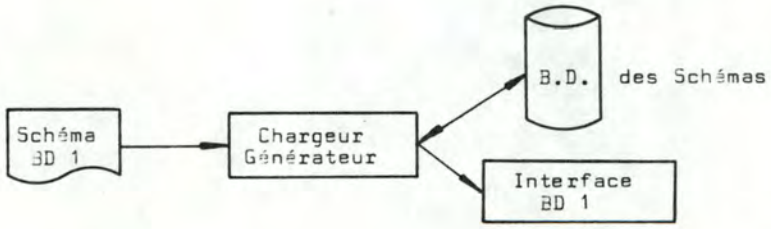


Fig. II.2. - Le sous-système de traitement des schémas.

Respect des objectifs.

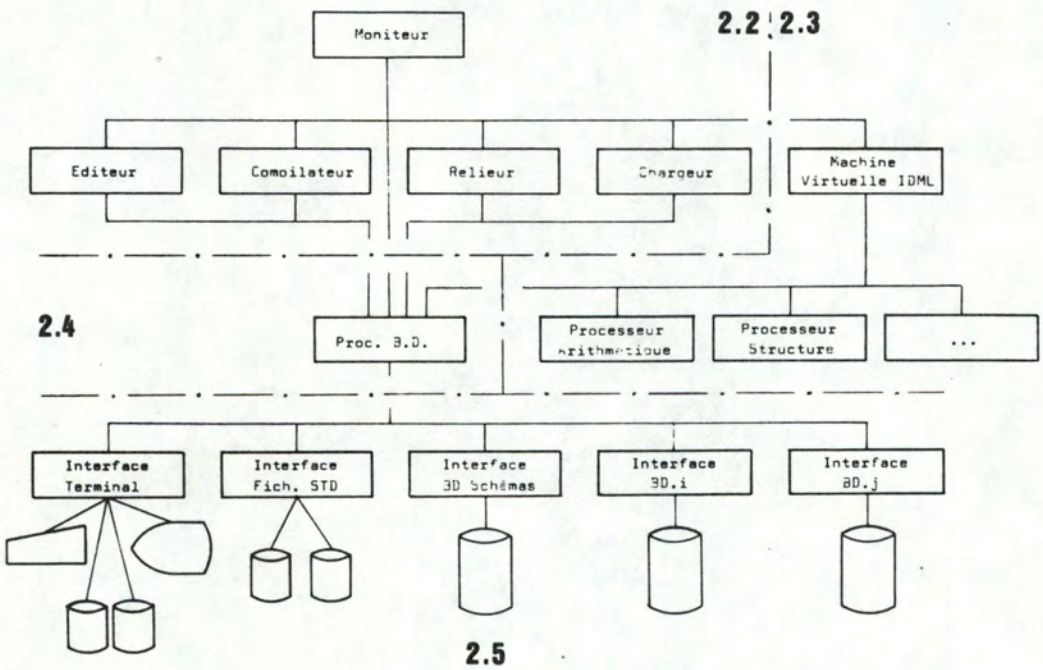
L'indépendance du système IDML est assurée par la généralisation des interfaces d'accès dont la génération est automatisée et par les modules spécialisés en nombre réduit. L'extensibilité est favorisée par la décomposition du système en modules distincts, ce qui est particulièrement vrai pour la machine virtuelle décomposée en processeurs spécialisés, permettant une extension ou une modification fonctionnelle de celle-ci.
Quant à la généralité, elle est liée essentiellement à la notion d'interface d'accès et à la structure du code intermédiaire.

Remarque générale portant sur les chapitres suivants.

Ce travail se base sur les spécifications du système IDML. Nous en avons retiré les éléments indispensables à la réalisation de l'objectif. Nous n'avons donc retenu qu'un sous-ensemble cohérent des spécifications.

En outre, certaines hypothèses d'implémentation seront énoncées mais sans justification quand elles sortent du cadre de recherche.

Nous allons analyser successivement les différents modules qui composent l'architecture de traitement des programmes IDML.



2.2. IMPLEMENTATION DES PROGRAMMES IDML.

Introduction.

Le langage de manipulation de données IDML permet, tant au programmeur qu'à l'utilisateur non spécialisé, de travailler de manière interactive sur une ou plusieurs Bases de Données.
Vu que par la suite nous ne traiterons plus ce langage de haut niveau, nous nous limiterons à introduire les objectifs et les concepts de ce langage (paragraphe 2.2.1.)

Par contre, nous nous intéresserons au cycle de traitement d'un programme IDML: compilation (paragraphe 2.2.3), assemblage (paragraphe 2.2.4.), chargement (paragraphe 2.2.5) et exécution (paragraphe 2.2.6.).

Mais afin de pouvoir comprendre chacune de ces phases, il est nécessaire d'analyser l'architecture de la Machine Virtuelle et de poser les principes de base du langage interne (paragraphe 2.2.2.)

2.2.1. Le langage IDML

Le langage de manipulation de données s'est vu assigner deux objectifs : la simplicité d'usage et d'expression.

La simplicité a été cherchée dans plusieurs directions :

- les données de toute Base de Données sont perçues selon les concepts du modèle d'accès, complet ou partiel et non ceux du SGBD;
- les mécanismes de programmation sont réduits mais extrêmement généraux;
- la sélection d'articles s'exprime sous des formes concises, intuitives et puissantes;
- dans de nombreux cas, certains raccourcis d'écriture conduisent à des expressions non algorithmiques.

La puissance et l'efficacité sont offerts au programmeur au travers :

- d'un langage de type algorithmique doté de mécanismes tels que la procédure (récursive), les manipulations de valeurs numériques et de caractères,...
- de l'exploitation (facultative) de la connaissance de certains mécanismes d'accès (par exemple la clé d'accès), de la manipulation aisée de collections d'articles.

Nous ne développerons pas le langage IDML lui-même. Citons cependant deux principes importants:

1. Une procédure IDML peut utiliser des variables déclarées explicitement ou implicitement;

2. la boucle IDML "FOR...END" constitue l'unique moyen d'accès aux Bases de Données

La condition de sélection d'articles peut prendre une des trois formes suivantes :

- condition sur les valeurs d'items associés;
- condition d'association à d'autres articles par chemin;
- condition d'appartenance à un ensemble.

Nous terminerons par un exemple :

- Le modèle d'accès de la Base de Données BDPERS :

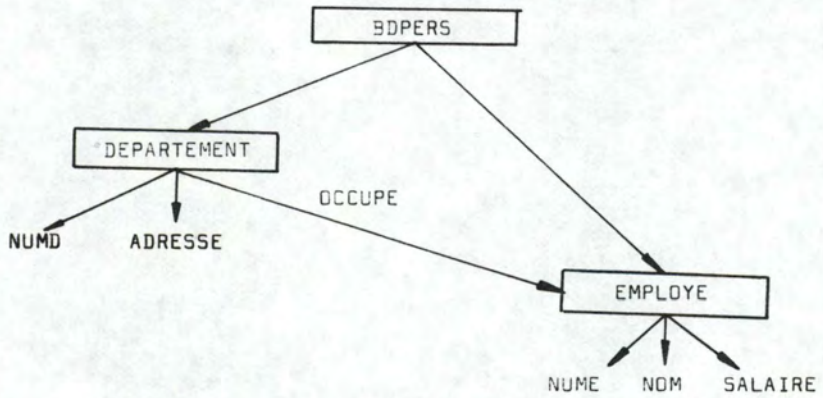


Fig. II.3 - Représentation graphique.

- Programme IDML :

afficher le nom de tous les employés dont le salaire est supérieur à 5000 frs et occupés dans un département :

```
for  B   :=  BDPERS
     for  DEP  := DEPARTEMENT
          for    E := EMPLOYE (salaire > 5000)
                      from DEP via OCCUPE
                 display E.NOM
          end
     end
end
```

2.2.2. La Machine Virtuelle IDML.

La Machine Virtuelle IDML est un élément essentiel du système IDML. De son architecture et de ses principes de fonctionnement, se dégagent la structure et la fonction des autres modules (compilateur, assembleur, chargeur).

2.2.2.1. Architecture de la Machine Virtuelle IDML

La Machine Virtuelle IDML est implémentée comme un interprète du langage interne.
Elle travaille donc sur des programmes écrits en langage interne. Pour ce faire, elle a à sa disposition un espace mémoire, deux registres particuliers et des processeurs spécialisés.

L'espace mémoire.

Cet espace mémoire est constitué d'un ensemble de mots standard, sous le contrôle total de l'interprète.
Cet espace est occupé d'une part par les instructions d'un programme; d'autre part par tout ce qui a rapport aux variables, c'est-à-dire les descripteurs et les valeurs des variables.

Pour pouvoir gérer cet espace, l'interprète dispose de deux informations :

Deux registres particuliers:

- le compteur ordinal (PC, Program Counter), qui contient l'adresse de l'instruction courante à exécuter;
- l'adresse du bloc des descripteurs de l'occurence courante de procédure (CDBA, Current Descriptors Block Address).

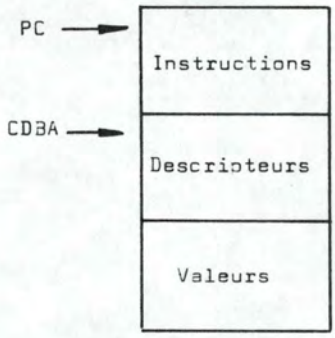


Fig. II.4 - L'espace mémoire et ses registres.

Les processeurs spécialisés.

Chacun de ces processeurs a pour rôle l'exécution d'une instruction demandée par l'interprète. Ces instructions ont été rassemblées en famille, chaque processeur spécialisé ayant une de ces familles à charge. Citons le processeur arithmétique, le processeur de transfert, le processeur de branchement, le processeur de structure et le processeur de Base de Données qui contrôle les différents interfaces d'accès aux Bases de Données. Ils seront développés dans le chapitre 2.3.
(- Spécification de l'interprète et de ses processeurs -)

2.2.2.2. Les concepts de base du langage interne

L'unité de construction (et de compilation) est la procédure.
Une procédure IDML est compilée en un module objet exprimé dans le langage interne. Un module objet est chargé dans la mémoire centrale de la Machine Virtuelle; une portion d'espace est occupée par les instructions d'une part, par le bloc de descripteurs et les valeurs d'autre part.

Un programme est constitué d'une procédure principale et des procédures susceptibles d'être appelées durant son exécution. Un programme objet contient un exemplaire du code objet de chacune de ces procédures.

A un instant déterminé de l'exécution d'un programme, une seule procédure s'exécute réellement; celle-ci est appelée la procédure courante.

Citons pour information qu'un appel à une procédure déjà active sera nommé "appel récursif".
Pour distinguer ces différents appels, nous associerons à chaque appel de procédure un processus appelé "occurence de la procédure".

Sans entrer dans trop de détails, il reste à voir la signification du descripteur d'une variable et à analyser l'adressage en mémoire centrale, afin de pouvoir saisir par la suite le rôle du compilateur et de l'assembleur.

Remarque. Par la suite nous n'avons implémenté que le cas où le programme est constitué d'une procédure unique. Nous ne parlerons dès lors de l'existence de plusieurs procédures que là où c'est nécessaire pour une bonne compréhension.

Une variable est caractérisée par :

- un type qui définit l'ensemble des opérations permises;

- une structure qui en définit la position dans une décomposition arborescente;

- un statut de protection qui précise en particulier la possibilité d'en modifier la valeur;

- une valeur qui est contenue dans un espace défini, par l'adresse de son début en mémoire centrale et sa longueur exprimée dans une mesure adéquate.

Certaines instructions font référence à une ou plusieurs valeurs de la mémoire centrale. Si l'on excepte pour l'instant les valeurs qui sont contenues dans l'instruction elle-même (mode d'adressage immédiat), se pose le problème du mode de référence de la valeur d'une variable par une instruction. C'est l'adressage absolu qui a été choisi : une valeur est désignée par l'adresse dans une table d'indirection de l'adresse absolue de cette valeur. En fait, chaque entrée de cette table contient la description de cette valeur : type, protection, structure, adresse et longueur. L'entrée dans la table est appelée descripteur.

La référence à une valeur par une instruction est réalisée par la désignation du descripteur de cette valeur.
Cette désignation est l'adresse de ce descripteur dans le bloc des descripteurs de la procédure (descripteurs représentés en longueur variable). Il s'agit donc d'un adressage relatif qui nécessite la connaissance de l'adresse d'implantation du bloc des descripteurs. Cette adresse n'est connue qu'à partir de la phase d'assemblage. De manière à conserver l'avantage de l'adressage indirect, l'adresse réelle du descripteur sera calculée lors de chaque référence à la valeur pendant l'exécution du programme.
En outre, les ordres de branchement sont relatifs à la valeur du compteur ordinal.

En résumé, au moment de l'exécution

- une instruction fait référence à une valeur par l'adresse relative du descripteur dans le bloc des descripteurs;

- un descripteur contient l'adresse absolue de la valeur en mémoire centrale;

- un ordre de branchement est relatif au compteur ordinal.

Le code d'un programme est relogeable.

2.2.3. Le moniteur

Le rôle du moniteur est de diriger et de contrôler le déroulement des différentes phases du traitement d'un programme.

En effet, l'utilisateur a à sa disposition un certain nombre de commandes. Par l'intermédiaire du terminal, il adresse une commande au Moniteur qui va se charger du suivi de son exécution : il lance le(s) module(s) concerné(s) et supervise, si nécessaire, l'enchaînement de ces modules.

Les commandes IDML sont les suivantes :

- un ensemble de commande pour l'édition qui ont pour but de créer ou de modifier un programme source, qui sera stocké dans un fichier standard de nom ‹nom-fichier›.S;

- COMP : ordre de compilation du programme source courant, qui fournira un fichier de nom ‹nom-fichier›.O;

- LINK : ordre d'assemblage du programme objet courant, dont le résultat sera placé dans un fichier de nom ‹nom-fichier›.E

- LOAD : ordre de chargement du programme exécutable courant;

- ‹nom-fichier› : exécution de la version la plus récente du programme contenu dans le fichier ‹nom-fichier›.S;

- ‹nom-fichier›.‹extension› : exécution de la version la plus récente du programme contenu dans le fichier ‹nom-fichier›.‹extension›;

  Les extensions retenues pour les fichiers sont :

  - S fichier contenant un texte source;
  - O fichier contenant du code objet;
  - E fichier contenant du code exécutable.

- END : ordre d'arrêt.

2.2.4. Compilation d'un programme.

La compilation d'un programme crée les blocs de données suivants:

- le bloc des instructions,
- le bloc des descripteurs,
- les valeurs initiales.

Rappelons :

- la référence à une valeur par une instruction est réalisée par la désignation du descripteur de cette valeur. Cette désignation est l'adresse relative du descripteur dans le bloc des descripteurs;

- les ordres de branchements sont relatifs au PC dans le bloc des instructions;
- le descripteur contient l'adresse relative de la valeur dans l'espace de mémoire réservé.

2.2.5. Assemblage d'un programme.

L'assemblage consiste à concaténer au bloc des instructions et des descripteurs de la procédure principale les blocs de toutes les procédures qui seront appelées durant son exécution, et à tranformer en adresses absolues les adresses présentes dans les descripteurs.

Une troisième fonction de l'assemblage est de résoudre les références aux procédures (appels). Une table des procédures sera construite dans ce but et reprendra, pour chaque procédure : son nom, l'adresse absolue de la première instruction et l'adresse absolue du bloc des descripteurs.

Le résultat de la compilation suivie d'un assemblage est un module stocké dans un fichier exécutable, sous la forme d'une liste de segments.

Un segment est constitué d'un descripteur de segment et d'un ensemble de valeurs.

Quatre types de segments ont été retenus :

1. instructions (segment relogeable, adresses relatives)
2. descripteurs (segment non relogeable, adresse absolue des valeurs en mémoire)
3. valeurs initiales
4. zone de travail libre .

Structure d'un descripteur de segments.

- type de segment
- adresse d'implantation en mémoire centrale
- longueur à charger (en nombre de mots)
- longueur occupée en mémoire centrale (en nombre de mots)

La longueur à charger est nulle quand le segment se rapporte à une zone de travail libre, mais la longueur occupée en mémoire libre représente la longueur d'un espace mémoire libre à réserver.

2.2.6. Chargement d'un programme.

Le chargement consiste à ranger dans la mémoire centrale, aux adresses absolues prévues lors de l'assemblage, les instructions, les descripteurs et les valeurs initiales.
Cette phase consiste à interpréter la liste des segments fournie par l'assemblage.

Il fournit au PC (compteur ordinal, contenant l'adresse de la prochaine instruction à exécuter) l'adresse absolue de la première instruction à exécuter. Cette adresse est connue du chargeur grâce au descripteur général du programme exécutable.

Outre cette adresse, le descripteur général renseigne le chargeur quant

- au type du programme (programme source, objet, exécutable);
- au numéro de version;
- au nombre de segments.

Il fournit au CDBA (Current Descriptors Block Address) l'adresse absolue d'implantation du bloc des descripteurs de la procédure principale.

2.2.7. Exécution d'un programme.

L'interprète exécute, à un instant déterminé, une instruction. L'adresse de cette instruction lui est donnée dans le PC.

A chaque accès à une valeur, il calcule l'adresse absolue du descripteur de cette valeur : adresse relative dans le bloc des descripteurs, augmentée de CDBA.

La valeur initiale du PC et la valeur de CDBA lui sont fournies lors du chargement du programme.

2.2.8. Exemple des résultats fournis par la compilation et l'assemblage de deux procédures.

Soient deux procédures IDML (a) et (b).

Chaque procédure est compilée séparément et fournit les informations suivantes :

La procédure (a) occupe n mots pour ses instructions, m mots pour les descripteurs et k mots pour les valeurs.

De même pour la procédure (b), on obtient N mots pour les instructions , M mots pour les descripteurs et K mots pour les valeurs.

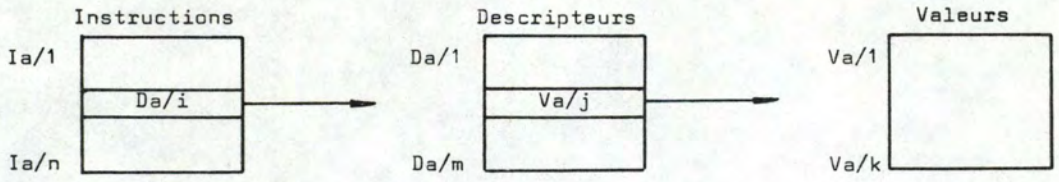


Fig. II.5. - Résultat de la compilation de (a).

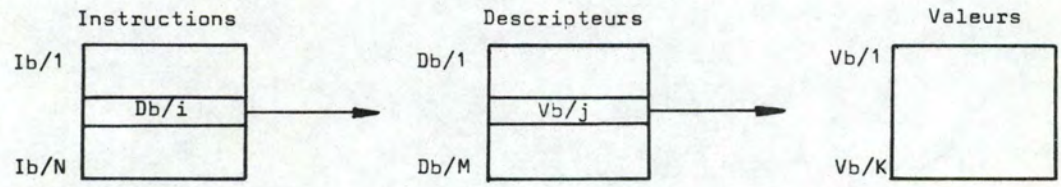


Fig. II.6. - Résultat de la compilation de (b).

Supposons de plus que (a) soit la procédure principale.

On obtiendra après l'assemblage les informations suivantes :
(voir Fig. II.7.)

- le PC vaut Ad 1
- la variable CDBA vaut Ad 3
- la table des procédures comprend la procédure de nom (b), dont la première instruction est à l'adresse Ad 2 et dont le bloc de descripteurs commence à l'adresse Ad 5.

L'assemblage a donc transformé

1. les références [Va/1 .. Va/k] en adresses [Ad 4 .. Ad 5]
2. les références [Vb/1 .. Vb/K] en adresses [Ad 6 .. Ad 7]

Enfin, lors de l'exécution, nous observerons les changements suivants:

1. les références [Da/1 .. Da/m] en adresses [Ad 3 .. Ad 4]
2. les références [Db/1 .. Db/M] en adresses [Ad 5 .. Ad 6]

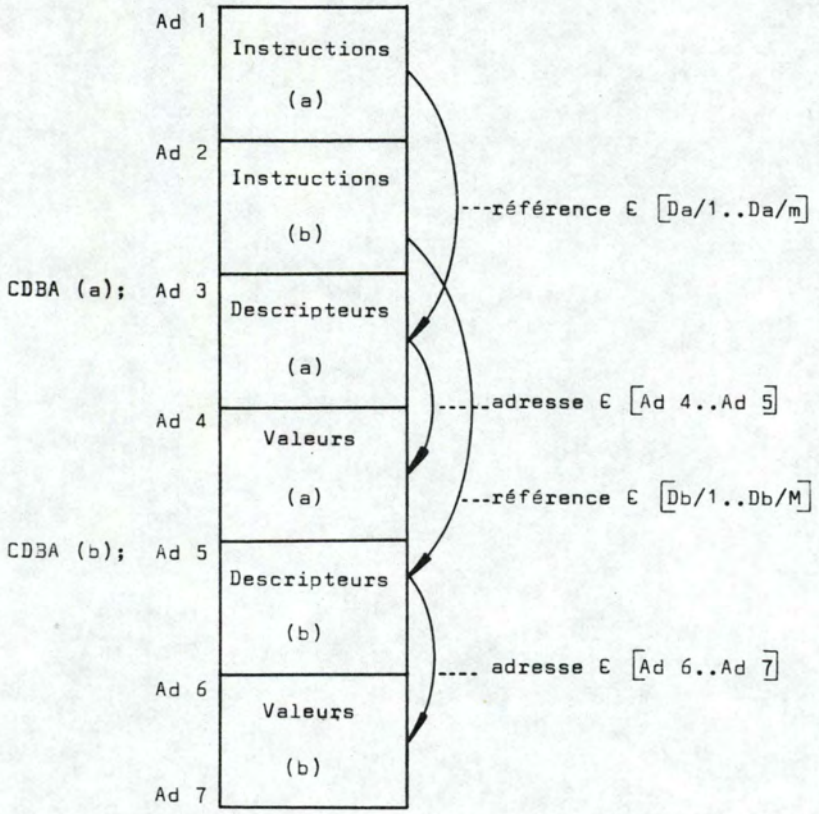


Fig. II.7. - Résultat de l'assemblage de (a) et (b).

2.3. SPECIFICATION DE L'INTERPRETE ET DE SES PROCESSEURS.

Introduction

Les deux éléments qui caractérisent le langage interne du système IDML sont les variables d'une part et les instructions d'autre part.

Tout d'abord nous traiterons des variables , et ce en quatre étapes (paragraphe 2.3.1.) :

- une classification des différentes variables;
- leur implémentation;
- la représentation en mémoire des descripteurs,
- et celle des valeurs.

Ensuite, nous nous intéresserons aux différentes instructions , et par là aux processeurs spécialisés (paragraphe 2.3.2.) :

- la représentation en mémoire des instructions (formats possibles);
- la structure du code opération;
- la syntaxe des instructions.

2.3.1. Les variables dans le langage de base.

2.3.1.1. Classification des variables.

Les variables sont classées selon deux critères : la structure interne d'une valeur et l'organisation de l'ensemble des valeurs d'une variable.

1. Organisation de l'ensemble des valeurs.

Les variables simples:

A une variable simple n'est associée qu'une seule valeur à un instant déterminé.
Si la variable est un composant, il existe une seule valeur par valeur de la variable dont elle est un composant.

Les variable répétitives:

A une variable répétitive sont associées n valeurs, n étant défini statiquement lors de la déclaration de la variable. Si la variable est un composant, il existe un tableau par valeur de la variable dont elle est un composant.

2. Structure des valeurs d'une variable.

Cette structure est celle d'une valeur de la variable. Nous retiendrons quatre grands types :

- la chaîne de caractères:

  elle est en longueur fixe, définie statiquement.
  Cette chaîne peut être de trois natures différentes :
  - les caractères appartiennent à l'alphabet général;
  - la chaîne est définie sur un alphabet restreint: on parlera de chaîne numérique;
  - la chaîne est numérique fixe: c'est une chaîne numérique avec marque décimale.

- la valeur numérique

  Différents formats ont été retenus :
  - l'entier en simple ou double précision;
  - le réel fixe, en simple ou double précision;
  - le réel flottant.

- la variable d'article:

  chaque valeur est une référence à un article d'une Base de Données.

- la valeur décomposable (ou structurée):

  une valeur résulte de la concaténation de valeurs de variables simples ou répétitives, décomposables ou non.

Les variables de ces quatre types peuvent être ou non des composants d'une variable décomposable. Elles peuvent en outre être simples ou répétitives.

2.3.1.2. Implémentation des variables.

1. Les variables simples non composants

chaîne de caractères :

- descripteur : type, adresse, longueur
- espace : une zone (adresse, longueur)

variable numérique :

- descripteur : type, adresse, format
- espace : une zone (adresse, longueur définie par format)

variable d'article :

- descripteur : type, adresse
- espace : une zone (adresse, longueur définie par type)
- valeur : référence de la BD et type de l'article

variable décomposable :

- descripteur : type, adresse, longueur
- espace : une zone (adresse, longueur)
- valeur : concaténation des valeurs des composants

2. Les variables répétitives non composants.

Une variable répétitive sera réalisée sous la forme d'une structure exprimant que l'élément est un composant du tableau.
Sa description précise sera donc abordée ultérieurement.

A ce niveau (variable non composant) un tableau est décrit comme une chaîne de caractères : il sera donc décrit par un type, une adresse et une longueur.
Un élément sera décrit (voir paragraphe suivant) par un type, l'adresse de l'un de ses éléments, la longueur de chaque élément le nombre de ceux-ci (répétition), ainsi que la référence du tableau (père).

Cette technique de représentation présente l'avantage suivant : après mise à jour de l'adresse d'un élément celui-ci est accessibles via son descripteur, comme une valeur simple.

Les valeurs sont rangées de façon contigüe en mémoire dans une zone (adresse du tableau, répétition * longueur d'un élément).

3. Les variables composants.

Une telle variable, simple ou répétitive est déclarée dans la description d'une variable décomposable.
Une variable décomposable est décrite complètement par un arbre de structure dont chaque noeud décrit une variable. A la racine correspond la variable non composant, il y correspond un descripteur du type vu dans le paragraphe précédent.
Quant aux autres noeuds, ils seront décrits par des descripteurs qui doivent permettre l'accès à une valeur particulière de la valeur associée.

La solution proposée est de définir sur l'ensemble des descripteurs une structure d'arbre décrivant la structure de décomposition des variables.

Le descripteur d'une variable composant contient un pointeur vers le descripteur de la variable dont elle est un composant (accès au père).

D'autre part, à la suite du descripteur d'une variable, seront rangés les descripteurs de ses composants, dans l'ordre dynastique. Cette implémentation permet donc tous les accès possibles avec un minimum de place supplémentaire.

Le déplacement d'un composant sera défini comme l'adresse relative de la première valeur de la variable dans chaque valeur de la variable père. L'accès à une valeur demande donc la connaissance préalable de l'adresse de cette valeur du père.

Le champ d'adresse contiendra l'adresse absolue de la valeur ou de l'une des valeurs s'il y en a plusieurs. Dans ce cas, ce champ est mis à jour par une primitive de calcul d'adresse.

En résumé, à une variable composant est associé un descripteur qui contient les informations relatives à cette variable si elle n'avait pas été un composant, ainsi qu'un pointeur père vers le descripteur de la variable dont elle est un composant et un déplacement qui est l'adresse relative dans la valeur "père".

Implémentation des tableaux.

Un tableau non composant est vu comme une structure à deux niveaux:

- le premier niveau décrit le tableau lui-même; il lui correspond un descripteur qui contient l'adresse de l'espace des valeurs du tableau, ainsi que sa longueur;

- le second niveau décrit chaque élément; il y correspond un descripteur qui contient, outre le type et la longueur de l'élément, un déplacement nul, une répétition, l'adresse d'un élément quelconque et l'information père référant au premier niveau.

A un tableau composant est associé un niveau supérieur, d'où une structure à trois niveaux.

2.3.1.3. Représentation des descripteurs.

1. Format des descripteurs.

Pour faciliter la compatibilité avec différentes machines, la longueur du mot standard est de 32 bits.

Un descripteur est constitué de un à trois mots standard. Des cinq types de mots retenus dans IDML, quatre occupent actuellement notre attention :

- M1 : type (12 bits) + adresse (20 bits)
- M2 : longueur (16 bits) + référence père (16 bits)
- M3 : répétition (16 bits) + déplacement (16 bits)
- M5 : refer-bd (8 bits) + type-article (8 bits)
  + référence père (16 bits)

Descripteur d'une variable simple non composant.

- chaîne de caractères : M1 , M2
- variable numérique : M1 , M2
- variable d'article : M1 , M5
- variable composée : M1 , M2

Descripteur d'une variable simple composant

- descripteur de la variable simple + M3

Descripteur d'une variable répétitive non composant

- tableau : M1 , M2
- élément : variable simple composant

Descripteur d'une variable répétitive composant

- tableau : M1 , M2 , M3
- élément : variable simple composant

2. Contenu du descripteur.

Le type de la variable.

Le type d'une variable est donné par la combinaison de cinq champs.

- la structure externe (STR-EXT) indique s'il s'agit d'une variable simple ou d'une table;
- l'état (STATUS) d'une variable est d'être réelle (par opposition à une variable fictive, propriété ignorée ici);
- la protection (PROTECT) accordée ou non à une variable fait d'elle une constante ou non;
- la structure interne (STR-INT) reprend les différents types de valeur d'une variable;
- l'unité (UNITE) indique l'unité de base de la zone mémoire occupée par la valeur de la variable.

Voici un tableau récapitulatif :

	STR-EXT (2)	STATUS (1)	PROTECT (1)	STR-INT (4)	(1)	UNITE (3)
valeur 0	var. simple	var. réelle	non protégée	caractère		-
1	table	-	protégée	car. numérique		-
2	-	-	-	car. num. fixe		-
3	-	-	-	entier		7 bits (D-7)
4	-	-	-	réel fixe		byte (D-9)
5	-	-	-	réel flottant		mot (8 bits)
6	-	-	-	article		
7	-	-	-	val. composée		

Fig. II.8. - Décomposition du type d'une variable.

L'adresse:

le champ adresse, qui occupe 20 bits, se décompose en deux éléments :

- l'adresse du mot standard en mémoire (16 bits);
- un déplacement de caractère (4 bits)

[Le contenu de DEPL-CAR est le déplacement en nombre de caractères de taille unité à partir de la plus proche frontière du mot.
Ce champ n'a de signification que si le "mot" peut contenir plus d'un caractère. Ce n'est pas le cas ici : DEPL-CAR sera toujours égal à zéro].

La longueur:

A nouveau, le champ longueur (16 bits) se scinde en deux éléments (de 8 bits chacun) :

- NBTOT indique le nombre d'UNITE occupées par la variable.
  Ainsi, pour un entier ou un réel fixe, cet élément donnera la précision :
  - NBTOT = 2 simple précision
  - NBTOT = 4 double précision
- NBDEC permet de coder la marque décimale si nécessaire (quand la structure externe de la variable vaut 2 ou 4).

A noter que pour une variable article la longueur est implicitement 4 mots (soit 32 bits).

2.3.1.4. Représentation des valeurs

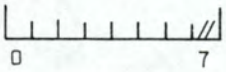
Nous avons vu que pour les instructions et les descripteursm le mot standard occupe 32 bits.

En ce qui concerne les valeurs, le mot de base est l'octet.

1. Représentation d'une chaîne de caractères.

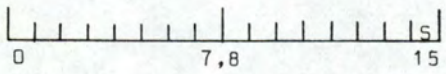
   Le code de caractère retenu est le code ASCII.
   Ce code nécessite 7 bits.

   Nous rangerons dès lors un caractère par mot, le dernier bit n'étant pas employé.

   le caractère : 

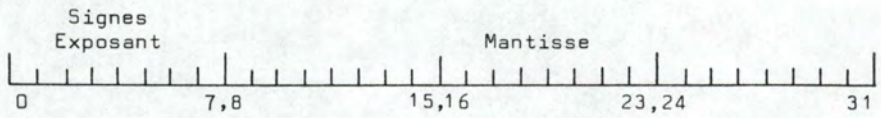
2. Représentation des valeurs numériques.

l'entier se range sur deux mots :



le réel flottant occupe quatre mots :

- le signe de la mantisse, le signe de l'exposant et l'exposant (les 8 bits inférieurs)
- la mantisse (les 21 bits supérieurs)



2.3.2. Les instructions du langage de base.

2.3.2.1. Format des instructions.

Ces formats sont représentés par groupe de mots de 16 bits, sans oublier que la taille standard du mot est 32 bits.

Les abréviations suivantes sont utilisées :

- COP : code opération
- DD : déplacement descripteur (dans la table)
- DI : déplacement instruction
- V : valeur

F1A	COP	/////							
F1B	COP	DD							
F1C	COP	DI							
F2A	COP	DD1	/////	DD2					
F2B	COP	DD	/////	V					
F2D	COP	DI	DD1	DD2					
F2E	COP	DI	DD	V					
F2F	COP	DI	/////	V					
F3	COP	DD1	DD2	DD3	DD4	DD5			

Fig. II.9. - Format des instructions.

2.3.2.2. Structure du code opération

Le code opération occupe 16 bits en mémoire.

Il est structuré en deux groupes :
- le numéro du processeur spécialisé (8 bits)
- le code de l'instruction elle-même dans le processeur en question (8 bits)

Ce code d'instruction est découpé en deux parties :
- 4 bits codifient la famille d'instruction
- 4 bits codifient l'instruction elle-même dans la famille.

Quant aux processeurs spécialisés, nous en avons distingués cinq :
- le processeur de transfert;
- le processeur arithmétique;
- le processeur de branchement;
- le processeur de structure;
- le processeur de base de données.

2.3.2.3. Les différents processeurs et leurs instructions.

Les codes d'instructions seront donnés sous leur forme symbolique et non numérique.
Trois abréviations sont utilisées :

- VAR : variable
- V : valeur immédiate
- DI : valeur de déplacement

1. Le processeur de transferts.

Quatre familles d'instructions ont été distinguées pour remplir les deux fonctions du processeur de tranfert :
- d'une part l'initialisation d'une variable;
- d'autre part, les opérations d'assignation.

- initialisation d'une variable - format F1B

```
MOV0  VAR    / init à zéro ou blanc
MOV1  VAR    / init à 1
MOVH  VAR    / init à la plus grande valeur
MOVL  VAR    / init à la plus petite valeur
```

- initialisation d'une variable d'article - format F2A

```
MOVA  VAR1, VAR 2
```

- assignation immédiate - format F2B

```
MOV1  VAR, V    / var := V
```

- assignation générale - format F2A

```
MOVE  VAR1, VAR2   / var1 := var2
```

2. Le processeur arithmèthique.

Comme son nom l'indique, ce processeur s'occupe des opèrations purement arithmètiques d'addition, soustraction, multiplication et division.

- arithmètique èlèmentaire - format F1B

```
INC  VAR     / var := var + 1
DEC  VAR     / var := var - 1
```

- arithmètique immèdiate - format F2B

```
ADDI  VAR, V     / var := var + V
SUBI  VAR, V     / var := var - V
MULI  VAR, V     / var := var * V
DIVI  VAR, V     / var := var / V
```

- arithmètique gènèrale - format F2A

```
ADD  VAR1, VAR2     / var1 := var1 + var2
SUB  VAR1, VAR2     / var1 := var1 - var2
MUL  VAR1, VAR2     / var1 := var1 * var2
DIV  VAR1, VAR2     / var1 := var1 / var2
```

3. Le processeur de branchement

Dans la plupart des cas, le dèroulement d'un programme se fait dans l'ordre dans lequel les opèrations sont rangèes en mèmoire.
Un type d'opèration particulier permet de rompre cette sèquence : le branchement vers une autre opèration.
Cette rupture peut être faite suite au rèsultat positif d'un test (branchement conditionnel) ou obligatoirement (branchement inconditionnel).

Rappelons que le dèplacement à effectuer est relatif à l'opèration courante.

- branchement conditionnel èlèmentaire sur code de retour - format F1C.

  Le code de retour est utilisè comme indicateur de la bonne ou mauvaise exècution d'une primitive.

```
BRZ     DI    / code_retour = 0 ? (pas d'erreur)
BRNZ    DI    / code_retour > 0 ? (erreur)
BRABS   DI    / article absent ?
BRNABS  DI    / article present ?

Si le test est positif alors PC := PC + DI
                       sinon PC := PC + 1;
```

- branchement conditionnel immédiat sur code de retour - format F2F.

```
BRLI    DI, V   / code_retour  <   V  ?
BRNLI   DI, V   / code_retour  >=  V  ?
BREI    DI, V   / code_retour  =   V  ?
BRNEI   DI, V   / code_retour  <>  V  ?
BRGI    DI, V   / code_retour  >   V  ?
BRNGI   DI, V   / code_retour  <=  V  ?

Si le test est positif alors PC := PC + DI
                        sinon PC := PC + 1;
```

- branchement conditionnel immédiat - format F2E.

```
BLI     DI, VAR, V   / var  <   V  ?
BNLI    DI, VAR, V   / var  >=  V  ?
BEI     DI, VAR, V   / var  =   V  ?
BNEI    DI, VAR, V   / var  <>  V  ?
BGI     DI, VAR, V   / var  >   V  ?
BNGI    DI, VAR, V   / var  <=  V  ?

Si le test est positif alors PC := PC + DI
                        sinon PC := PC + 1;
```

- branchement conditionnel général - format F2D

```
BL    DI, VAR1, VAR2   / var1  <   var2  ?
BNL   DI, VAR1, VAR2   / var1  >=  var2  ?
BE    DI, VAR1, VAR2   / var1  =   var2  ?
BNE   DI, VAR1, VAR2   / var1  <>  var2  ?
BG    DI, VAR1, VAR2   / var1  >   var2  ?
BNG   DI, VAR1, VAR2   / var1  <=  var2  ?

Si le test est positif alors PC := PC + DI
                        sinon PC := PC + 1;
```

- branchement inconditionnel

```
B   DI     / PC := PC + DI          format F1C
BV  VAR    / PC := PC + var         format F1B
```

- arrêt de l'exécution - format F1A

```
STOP
```

Quand le programme inclut plusieurs procédures, on a besoin de deux instructions supplémentaires :

- appel à une procédure,
- retour à la procédure appelante,

qui sont des branchements particuliers.

5. Le processeur de structure.

Ce processeur a pour objet la mise à jour de l'adresse d'une variable (adresse contenue dans la description).

Deux situations peuvent nécessiter une telle opération :

- pour adresser un élément composant non répétitif : son adresse est calculée sur base de l'adresse de la valeur père (la variable composée) à laquelle il faut ajouter le déplacement qui est indiqué dans son descripteur.

- pour adresser un élément répétitif particulier : l'adresse de l'élément de rang i est calculée comme étant l'adresse de la valeur père (le tableau) augmentée du déplacement éventuel et de (i-1) fois la longueur de l'élément.

- Adressage d'un élément répétitif. - format F2A

```
SETAD   VAR1, VAR2  /  VAR1 : l'élément répété
                    /  VAR2 : le rang de l'élément
```

- Adressage d'un élément composant non répétitif. - format F1B

```
SETA   VAR    /   VAR : l'élément composant
```

6. Le processeur Base de Données.

Ce processeur a pour fonction de permettre à l'utilisateur de travailler sur des Bases de Données.
Cela se fait par l'intermédiaire d'interfaces spécialisées.

Cette interface et les paramètres nécessaires seront expliqués au chapitre 2.5.
(- Les interfaces Base de Données -)

Citons l'instruction du langage qui permet ce travail:

- appel du processeur Base de Données - format F3

```
BDC   VAR1, VAR2, VAR3, VAR4, VAR5
```

2.4. UN PROCESSEUR PARTICULIER : LE PROCESSEUR BASE DE DONNEES.

Si nous avons réservé un chapitre particulier à ce processeur, c'est parce que sa fonction est primordiale dans le système IDML.

Car, si comme nous l'avons vu, ce processeur peut être appelé par une instruction du langage interne, il peut aussi être invoqué par tous les autres modules du système IDML.
En effet, nous avons dit que toute communication avec l'extérieur est perçue comme une transaction sur une Base de Données.

Le système IDML comporte, dès lors, plusieurs Bases de Données distinctes, chacune sous le contrôle d'une interface qui lui est propre :

- Base de Données Terminal
- Base de Données Fichiers Standard
- Base de Données conventionnelle.

Nous sommes dans un contexte multi-Bases de Données.

Toute demande d'une transaction (primitive) sur une Base de Données doit être adressée au processeur Base de Données.

Celui-ci a pour rôle de diriger cette demande vers l'interface, et donc vers la Base de Données concernée.

Afin de pouvoir établir la correspondance entre une transaction et une Base de Données particulière, une référence sera attribuée à chaque Base de Données et devra être mentionnée par l'utilisateur à chaque appel.

Cependant, au départ, l'utilisateur ignore la valeur de cette référence. C'est pourquoi, lors du premier appel (normalement une demande d'ouverture de la Base de Données), le processeur Base de Données interrogera successivement les différentes Bases de Données jusqu'au moment où une d'entre elles se sera reconnue, ou que les possibilités seront épuisées (c'est-à-dire que la Base de Données est inconnue).

Une fois la transaction exécutée, le processeur Base de Données rend la main au module qui l'a appelé.

2.5. LES INTERFACES BASE DE DONNEES.

Introduction

Nous avons déjà mentionné à plusieurs reprises le rôle joué par les interfaces Bases de Données.

Nous allons développer le contexte dans lequel elles travaillent (paragraphe 2.5.1.), puis énoncer les spécifications sur lesquelles elles reposent (paragraphe 2.5.2.)

Enfin, nous présenterons les interfaces associées aux deux Bases de Données non conventionnelles (la Base de Données Terminal et la Base de Données Fichiers Standard) (paragraphe 2.5.3.)

2.5.1. Le contexte.

Le mot interface désigne un programme, spécifique à une Base de Données chargé de réaliser toutes les requêtes émises par des utilisateurs, qui concernent cette Base de Données. Une requête consiste en une demande d'exécution d'une des primitives définies dans le modèle d'accès. (les concepts du modèle d'accès ont été définis au paragraphe 2.1.2. - Le modèle de Données -)

Le rôle de l'interface est de présenter à l'utilisateur un langage de manipulation de données unique, quel que soit le SGBD gérant la Base de Données.
L'interface exécute une requête (primitive du modèle d'accès) au moyen des ordres du langage de manipulation de données du SGBD gérant réellement la Base de Données.

L'interface est autonome : aucune contrainte n'est imposée sur l'environnement dans lequel elle est appelée à être utilisée : que l'on se trouve dans un contexte "multi-Bases de Données" ou "mono-Base de Données", l'interface et son utilisation sont inchangées.
Elle doit être capable de prendre en charge la gestion simultanée de plusieurs utilisateurs.

L'interface constitue le moyen de matérialiser l'avantage offert par un modèle unique de description des SGBD tel que le modèle d'accès.
Tout utilisateur familiarisé au modèle d'accès, sans connaître les particularités de tel ou tel SGBD, pourra, s'il dispose de la description en termes du modèle d'accès d'une Base de Données y accéder au travers d'un interface

2.5.2. Spécification générale de l'interface.

Une interface est invoquée par un appel; un appel correspond à une demande d'exécution d'une primitive.
Les paramètres et zones de transmission d'informations sont transmis au moment de l'appel à l'interface.

2.5.2.1. Notions particulières

Notions de références à un objet.

La référence à un objet est un identifiant associé automatiquement par l'interface, à tout objet auquel on a accédé avec succès et qui permet de le désigner lors de toute demande ultérieure le concernant.
Les objets susceptibles de se voir attribuer une référence sont les Bases de Données, les fichiers et les articles.

La référence Base de Données n'est pas attribuée par l'interface. Cette référence, qui permet d'identifier une Base de Données parmi d'autres, ne joue pleinement son rôle que dans un contexte multi-Bases de Données. Elle sera, dès lors, attribuée par le processeur Base de Données.

Durée de vie d'une référence à un objet.

Toute référence disparaît avec le processus durant l'exécution duquel elle a été attribuée.
Pendant l'exécution du processus, une référence attribuée ne le sera pas une seconde fois à un autre objet.

Utilisation de la référence à un article d'un fichier.

Les possibilités d'accéder à un objet sur base de sa référence, vont dépendre du SGBD-cible et des options prises lors de la spécification de l'interface.

Ainsi, l'option implémentée permet de réaliser l'accès par référence à un article du fichier, pour autant que ce soit le dernier article de ce fichier auquel il ait fait accès.
En d'autres termes, il faut que cette référence corresponde à la référence courante dans le fichier pour l'utilisateur en question.

Une ambiguïté se produit lorsqu'on procède à des suppressions d'articles, sur base de la référence d'article : l'article supprimé, sa référence n'en est plus une.
Si l'on veut accéder à l'article suivant celui supprimé, on considérera que la référence de l'article supprimé peut encore être utilisée comme référence de l'article précédent.

Notion de code de types d'objets.

Rappelons les différents types d'objets que nous avons retenus dans le modèle d'accès : la Base de Données, le fichier, le type d'article (et le type d'item).

Deux possibilités de désignation ont été choisies, soit le nom complet, soit un code.

Les deux possibilités ont été utilisées dans l'interface : la Base de Données et les fichiers seront désignés par leur nom, tandis que les types d'articles le seront par un code.
Ces noms et codes sont connus de l'interface, qui pourra ainsi faire les vérifications nécessaires.

En résumé :

	Désignation du type	Désignation de l'objet	Attribuée par
BD	nom	référence SREF	proc.BD
fichier	nom	référence RFIL	interface
type article	code	référence RREF	SGBD/SGF

Fig. II.10. - Désignation des types d'objets et des objets.

2.5.2.2. Les paramètres d'appel.

En toute généralité, cinq types particuliers de paramètres sont distingués :

- les paramètres précisant l'opération à effectuer et les objets sur lesquels elle s'applique :
  Z-CODES
- les paramètres permettant le transfert de valeurs d'identifiants ou de valeurs d'objets de l'utilisateur vers l'interface :
  Z-IDENT / Z-VALUES
- des paramètres pour la transmission d'une liste de codes d'items :
  Z-ITEMS
- des paramètres pour la réception de valeurs d'items
  RFIELD
- des paramètres concernant les chemins d'accès :
  Z-SETS

Nous présentons maintenant chacune de ces classes, en ne développant que les paramètres retenus pour les Bases de Données implémentées.
Rappelons que cette implémentation n'était pas le but du travail, c'est pourquoi elle a été réduite au minimum nécessaire pour pouvoir valablement travailler par la suite. De plus amples

informations sont disponibles dans l'annexe II. (- Les interfaces BD dans le système IDML -).

La taille des paramètres est standard, quelle que soit l'implémentation :

- Z-CODES occupe 30 octets,
- Z-IDENT a besoin de 90 octets,
- Z-VALUES ET RFIELD se rangent sur 256 octets.

1. Les arguments Z-CODES

- COP code opération

  un code opération a été attribué à chaque primitive. Lors de tout appel, COP doit contenir l'un de ces codes. Si tel n'est pas le cas, aucune primitive ne sera exécutée.

- SREF référence de la Base de Données

  cette référence qui identifie la Base de Données est fournie à l'interéface lors du premier appel (ouverture de la Base de Données); SREF devra contenir cette même référence lors de tous les appels successifs, sinon aucune action ne sera faite par l'interface.

- COREC code de type d'articles

  lors de la demande d'exécution d'une primitive qui requiert parmi ses paramètres une identification de type d'articles, COREC doit contenir le code numérique attribué au type d'article en question.

- RETCODE code de retour

  RETCODE contient, après l'exécution d'une requête par l'interface, le diagnostic de cette exécution (valeur 0 signifiant une exécution normale; toute autre valeur signifiant la détection d'une anomalie).

- PROTECT code de protection

  il existe deux interprétations de ce paramètre.

  - en ouverture de la Base de Données :
    PROTECT signale si le mode d'ouverture des fichiers est normal (on ne peut accéder à un article de la Base de Données que si on a ouvert préalablement le fichier qui le contient) ou automatique (c'est-à-dire que la notion de fichier est ignorée de l'utilisateur);

  - en ouverture de fichier :
    PROTECT permet de préciser la protection désirée (aucune, protégée, exclusive). On y associe aussi la notion d'intention de travail (consultation, mise-à-jour).

- COGET code d'accès aux valeurs d'items d'un article

  COGET permet, lors d'un accès à un article, de signaler à l'interface comment on désire accéder aux valeurs d'items de cet article : pas d'accès aux valeurs d'items; accès à toutes les valeurs d'items; accès aux valeurs d'items dont les codes des types se trouvent dans Z-ITEMS.

- RFIL référence de fichier

  RFIL est garni par l'interface lors de l'ouverture du fichier; toute requête ultérieure concernant ce fichier devra fournir cette référence.

- RREF référence d'article

  après un accès réussi à un article, RREF contient la référence de cet article; RREF doit être garni de cette référence pour l'appel de certaines primitives.

- PREF référence de l'article précédent

  PREF est utilisé pour l'accès aux articles; il doit contenir la référence du dernier article auquel on a accédé. Si PREF = 0, on accèdera au premier article (d'un type ou d'un fichier suivant la primitive).

Il existe encore d'autres paramètres dans cette classe :
- contrôle d'article;
- code clé d'accès
- opérateur;
- cible d'un type de chemin;
- référence d'article origine ou cible d'un chemin.

2. Les arguments Z-IDENT / Z-VALUES

- SSNAME nom de la base de données

  ce nom est à fournir lors de la demande d'ouverture de la Base de Données.

- PSW mot de passe

  mot de passe à fournir lors de la demande d'ouverture de la Base de Données. S'il ne correspond pas à celui défini pour la Base de Données gérée par l'interface, l'accès ne sera pas autorisé.

- FILNAME nom de fichier

  nom à fournir lors de la demande d'ouverture d'un fichier; ce nom doit être celui de l'un des fichiers appartenant à la Base de Données gérée par l'interface.

- Z-VALIT valeur des items d'un article

  destiné à contenir les valeurs d'items des articles lors de la création d'un article, ou de la modification de valeurs d'items d'un article.

En outre, le paramètre Z-CLE permet de fournir une valeur de clé.

3. L'argument Z-ITEMS

Il contient une liste de codes d'items.
Cette liste est utile pour permettre à l'utilisateur de n'accéder qu'à une partie d'un article, certaines valeurs d'items étant momentanément laissées de côté.
Cette zone n'est pas utilisée dans les interfaces présents.

4. L'argument RFIELD

RFIELD est destiné à recevoir les valeurs des items de l'article auquel vient d'accéder l'utilisateur.

5. Les arguments Z-SETS

représentent des paramètres tels que référence de stockage, liste de code de type de chemin d'accès, liste de références d'articles associés à des chemins.

2.5.2.3. Les primitives.

Nous n'aborderons pas ici la spécification détaillée de chacune des primitives qu'un interface est susceptibles de mettre à la disposition des utilisateurs.
Pour plus d'informations [paramètres en Entrée, paramètres en Sortie, diagnostiques d'erreurs], on se réferera à l'annexe II. (- Les interfaces BD dans le système IDML -).

1. Les primitives d'accès

- accès à la base de données :
  - ouverture de la Base de Données,
  - fermeture de la Base de Données.

- accès aux fichiers :
  - ouverture de tous les fichiers,
  - ouverture d'un fichier,
  - fermeture de tous les fichiers,
  - fermeture d'un fichier.

- accès aux articles :
  - accès séquentiel aux articles d'un type donné,
  - accès par clé aux articles d'un type donné,
  - accès séquentiel aux articles d'un fichier,
  - accès par clé aux articles d'un fichier,
  - accès séquentiel aux articles cibles d'un chemin,
  - accès par clé aux articles cibles d'un chemin,
  - accès par référence à un article.

2. Les primitives de modification.

- primitives ayant pour cible un fichier :
  - réinitialisation d'un fichier.

- primitives ayant pour cible un article :
  - création d'un article,
  - suppression d'un article,
  - modification des valeurs d'items d'un article,
  - insertion d'un article dans un chemin,
  - retrait d'un article d'un chemin.

3. Les primitives de contrôle :

- demande de contrôle sur un article (bloquage),
- demande de libération d'un article (débloquage),
- etc ...

2.5.3. Deux Bases de Données particulières.

Nous avons vu que toute communication avec l'extérieur est perçue comme une transaction sur une base de données.

Nous devons, dès lors, distinguer deux Bases de Données particulières :

1. la Base de Données "terminal"

2. la Base de Données "fichiers standard".

la première comporte le terminal du système et un fichier permettant de travailler en batch;
la seconde recouvre l'ensemble des fichiers de travail :
fichier programme-source, fichier programme-code, fichier programme-exécutable, fichier utilisateur (données, résultats).

Pour chacune de ces Bases de Données, l'ensemble des primitives accessibles est restreint. Il en est de même de l'ensemble des paramètres. L'implémentation est détaillée en annexe II.

2.5.3.1. L'interface Base de Données Terminal.

- Ouverture de la Base de Données.

  le nom de la Base de Données est "BDIO"

- Fermeture de la Base de Données.

- Ouverture d'un fichier.

  - le fichier 'TIN" correspond au clavier du terminal,
  - le fichier "TOUT" correspond à l'écran,
  - le fichier "BIN" est le fichier batch en entrée,
  - le fichier "BOUT" est le fichier batch en sortie.

Les fichiers TIN et BIN ne peuvent être ouverts qu'en lecture, tandis que les fichiers TOUT et BOUT ne sont accessibles qu'en écriture.

- Fermeture d'un fichier.

- Lecture d'un article.

la lecture d'un article ne peut se faire que sur le fichier TIN ou le fichier BIN.

Vu que le clavier (TIN) est un support purement séquentiel, seule une lecture séquentielle peut être implémentée.
C'est pourquoi l'interface ne tiendra pas compte des paramètres PREF (référence de l'article précédent) et RREF (référence de l'article courant).
Quelles que soient leurs valeurs, l'utilisateur accèdera au premier message disponible.

Par similitude, nous avons restreint la lecture sur le fichier batch BIN à un accès purement séquentiel.

Quant au paramètre COREC (code de type d'article), il prend ici une signification particulière : la longueur du message.

  - si COREC = 0 : lire jusque quand on trouve un caractère <CR> ou que 132 caractères ont été lus. La chaîne ainsi lue est renvoyée, terminée par un caractère <NULL>.

  - si COREC > 0 : lire la longueur spécifiée (avec un maximum de 132 caractères), et la renvoyer terminée par le caractère <NULL>.

- Ecriture d'un article.

Tout comme la lecture d'un article, l'écriture d'un article sur le fichier "TOUT" ou "BOUT" est une action essentiellement séquentielle.

COREC spécifie encore la longueur de l'article.

  - si COREC = 0 : écrire jusque quand on trouve un caractère <CR> ou que 132 caractères ont été écrits.

  - si COREC > 0 : écrire l'article de la longueur donnée.

2.5.3.2. L'interface Base de Données Fichiers Standard.

- Ouverture de la Base de Données.

le nom de la Base de Données est 'BDFI'.

- Fermeture de la Base de Données.

- Ouverture d'un fichier.

  la Base de Données Fichiers Standard est dynamique.

  Elle n'est pas constituée d'un ensemble déterminé de fichiers. C'est l'utilisateur qui, par l'ouverture d'un fichier, met ce fichier sous le contrôle de l'interface.
  Il s'agit essentiellement de fichiers de caractères séquentiels, ouverts en Lecture ou en Ecriture.
  L'interface permet la gestion simultanée de deux fichiers en lecture et deux fichiers en écriture.

- Fermeture d'un fichier.

- Lecture d'un article.

- Ecriture d'un article.

  La longueur de la chaîne à lire ou à écrire est donnée par le paramètre COREC, dont les valeurs correctes appartiennent à l'intervalle [0..256].

<CR> : caractère de code Ascii 13,
<NULL> : caractère de code Ascii 0.

P A R T I E 3. LE SYSTEME IDML PARTAGE.

3.1. PRINCIPES ET ARCHITECTURE.

Introduction.

Avant de présenter la structure du système partagé, il est bon d'en percevoir l'objectif et de le situer dans l'ensemble du travail (paragraphe 3.1.1.).

Puis, nous rappelerons les trois niveaux de dialogue que nous avons déjà distingués (paragraphe 3.1.2.) et ceci afin de construire l'architecture nécessaire pour chaque type de travail (paragraphe 3.1.3.)

Enfin, nous décrirons l'environnement dans lequel s'est placée la réalisation, le matériel hardware et le langage de programmation (paragraphe 3.1.4.).

3.1.1. Objectifs du système IDML partagé.

L'objectif premier de ce travail consiste à offrir à l'utilisateur un outil de manipulation de données; données qui seraient réparties (ou distribuées) sur plusieurs sites d'un réseau.

Chaque site sur le réseau peut remplir deux fonctions :

1. stocker des données et fournir des accès à ces données, tant pour un utilisateur local (c'est-à-dire sur le site même) que pour un utilisateur réseau.

2. permettre à l'utilisateur local d'accéder à la totalité des données disponibles, quel que soit le site de stockage.

D'un système réalisant ces deux fonctions, nous pourrons effectivement dire qu'il est le support de Bases de Données Réparties. Un système ne réalisant que la fonction de stockage et point d'accès constitue ce que nous appellerons un système partagé.

Ce système partagé consiste donc à rendre le système IDML partageable entre plusieurs utilisateurs, c'est-à-dire multi-utilisateurs. La difficulté provient de la diversité géographique de ces utilisateurs :

- utilisateur local : il est en communication directe avec le système;

- utilisateur réseau : toute communication avec le système passe par l'utilisation du réseau.

Mais afin de laisser au système IDML ses spécifications premières il faut absolument que cette diversité lui soit transparente. Quel que soit l'utilisateur, le système IDML travaille de la même façon.

C'est pourquoi, afin de prendre en charge ces différents utilisateurs, le système IDML mono-utilisateur a été étendu à un système partagé.

Voici l'architecture générale proposée. Nous n'entrerons pas ici dans l'explication et la spécification de cette structure. Ce point essentiel sera abordé dans les chapitres 3.3. (- Implémentation du niveau Primitives -) et 3.4. (- Implémentation du niveau Langage de Base -).

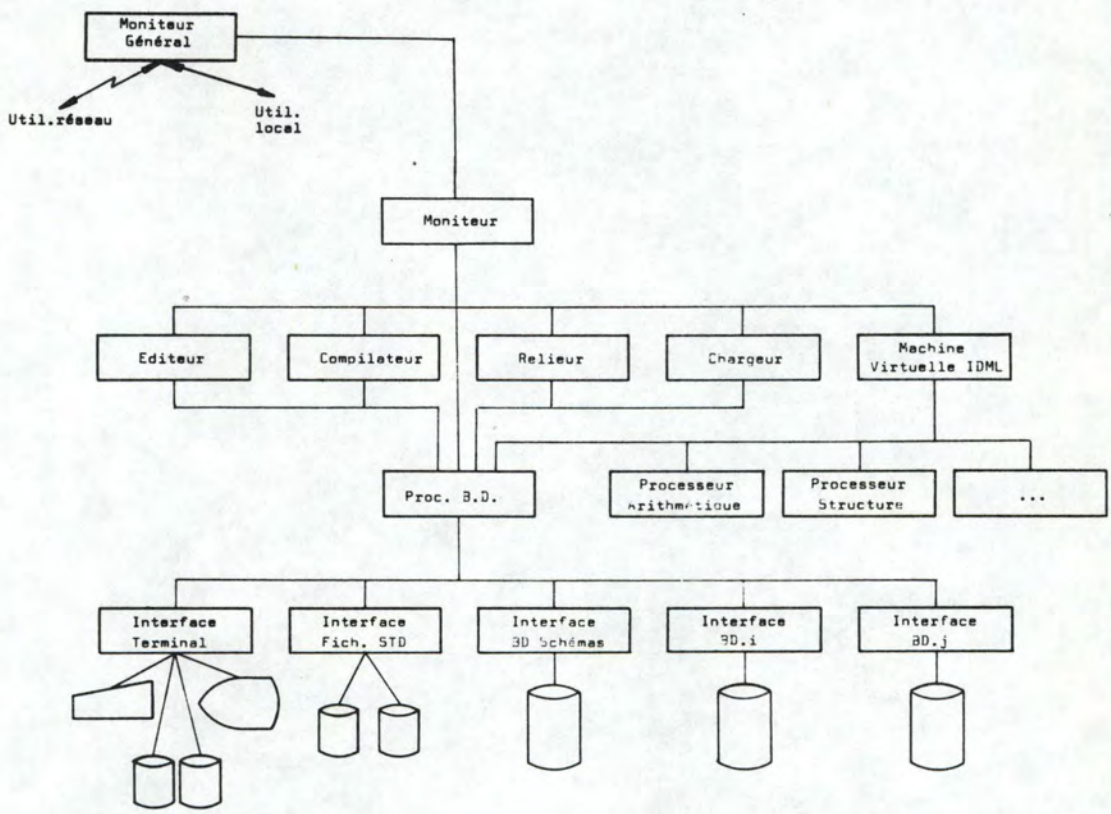


Fig. III.1. - Le système IDML partagé.

Justification de la démarche de travail.

La raison pour laquelle nous avons traité séparément le problème du système partagé du système distribué est double.

En premier lieu, il s'agissait de clarifier l'analyse du problème. Les difficultés à surmonter sont de nature différentes : dans le premier cas, il faut réaliser sur un site mono-tâche une application multi-tâches et gérer un système multi-utilisateurs;

dans le second cas, il faut résoudre des questions propres au contexte de Base de Données Répartie, telles que la localisation du dictionnaire des informations, le contrôle des données, l'analyse d'une requête...

De plus, il serait absurde de pouvoir émettre des demandes à d'autres sites si ces sites ne sont pas prêts à gérer ces requêtes externes.

Dès lors, il apparaît nettement que le système partagé est une étape indispensable vers la réalisation d'un système distribué.

Ensuite, cette architecture partagée peut être utilisée en dehors du contexte de Base de Données Répartie. En effet, elle permet l'accès à une Base de Données centralisée, mais dans un contexte d'informatique répartie.

3.1.2. Les trois niveaux de communication.

Nous avons vu au chapitre 1.2.4. (- L'interaction homme-machine : le langage de communication -) que le dialogue entre l'utilisateur et la Base de Données peut se situer à trois niveaux.

- expression de primitives,
- expression d'un programme en langage interne (Lgbs),
- expression de requêtes dans un langage de haut niveau (IDML).

Chaque utilisateur pouvant travailler au niveau qui lui convient, il faut que le système IDML partagé accepte ces trois formes de communication.

L'évaluation et l'intérêt de ces trois niveaux fera l'objet de la dernière partie.

L'implémentation du système partagé utilise des éléments de l'architecture, différents d'après le niveau de dialogue. Pour la clarté de l'étude, nous posons l'hypothèse de travail suivante : tous les utilisateurs travaillent au même niveau.

Mais il est évident que dans la réalité, l'implémentation permet à chaque utilisateur de dialoguer dans l'expression adéquate à son travail et à ses possibilités.

Avant de regarder précisément l'implémentation de ces différents niveaux, voyons l'architecture nécessaire à chaque forme de dialogue.

Lgbs : abréviation utilisée pour Langage de base du système IDML.

3.1.3. Les trois architectures correspondant aux trois niveaux.

1. Le niveau Primitive.

Le Moniteur Général est à l'écoute de l'utilisateur local et des utilisateurs réseau.

Chaque fois qu'une demande de traitement d'une primitive lui parvient, il transmet les paramètres au processeur de Base de Données (PROC. BD).

En outre, le Moniteur Général fournit au processeur Base de Données la référence interne de l'utilisateur, afin que les différentes interfaces Base de Données puissent gérer les Bases de Données multi-utilisateurs.
Les interfaces qui sont sous le contrôle du PROC. BD sont toutes les interfaces Base de Données conventionnelles.

Une fois le traitement de la primitive terminé, le PROC. BD transmet le résultat (l'ensemble des paramètres) au Moniteur Général qui se charge alors de les communiquer à l'utilisateur.

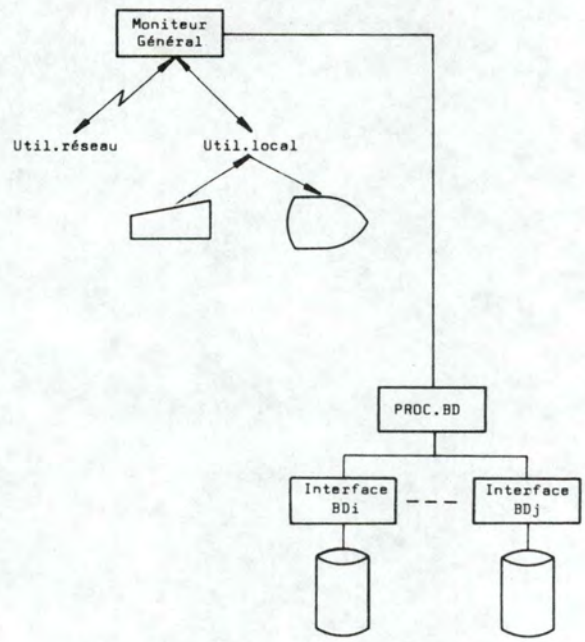


Fig. III.2. - Le système IDML partagé : niveau Primitives.

2. Le niveau Lgbs (langage de base, langage interne).

Le Moniteur Général est à l'écoute de l'utilisateur local et des utilisateurs réseau.

Quand l'utilisateur local signale qu'il désire travailler en Langage de base, le Moniteur Général passe directement la main au Moniteur qui travaille comme s'il était dans le contexte mono-utilisateur.

Il s'adressera à l'interface Terminal pour obtenir les commandes.

Il contrôlera les différents modules : le chargeur, qui travaille avec l'interface Base de Données des Fichiers Standard; l'interprète, qui peut travailler avec toutes les interfaces de Bases de Données (Terminal, Fichiers Standard, Conventionnelle) suivant les indications du programme interprété.

Tous les accès aux interfaces se font par l'intermédiaire du Processeur Base de Données.

Quand un utilisateur réseau signale qu'il désire travailler en Langage de base, le Moniteur général prend un rôle important.

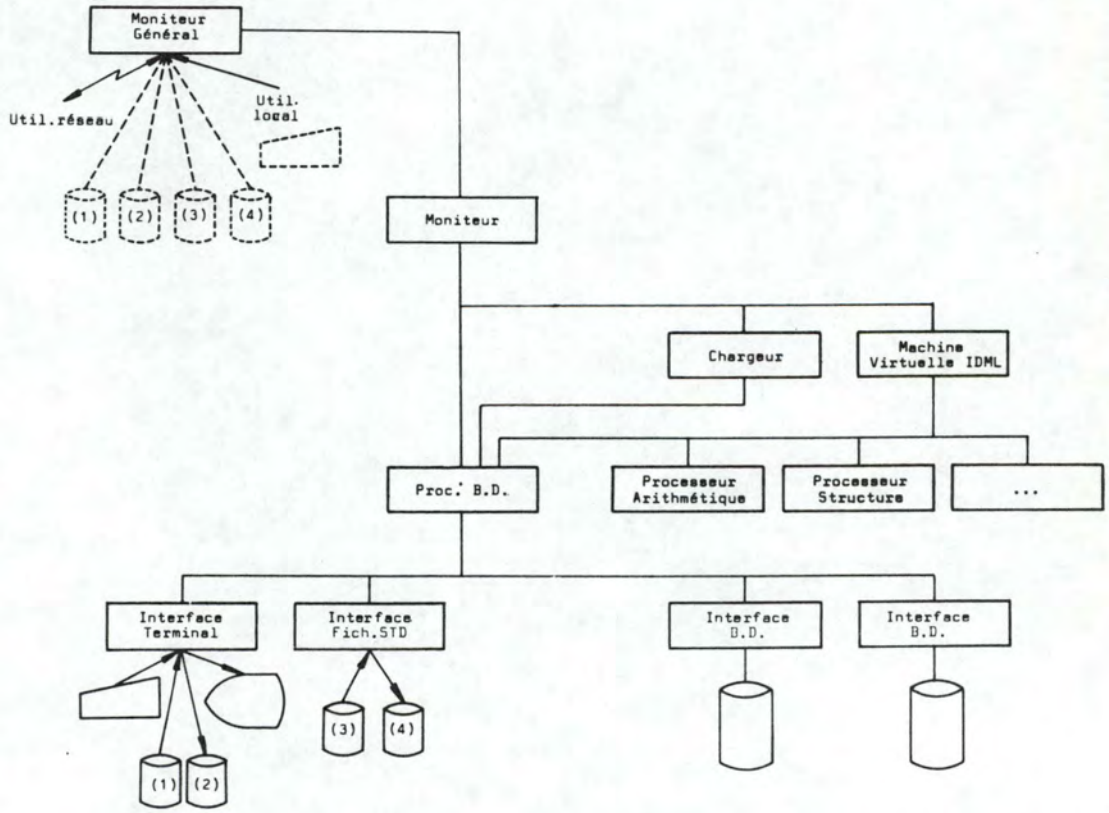


Fig. III.3. - Le système IDML partagé : niveau Lgbs.

- il réceptionne le programme en langage de base et le range en un fichier exécutable x.E (3).
  Ce fichier sera dès lors sous le contrôle de l'interface Fichiers Standard.

- il crée un fichier de commande terminal : demande d'exécution du programme contenu dans le fichier x.E.
  Ce fichier sera dès lors sous le contrôle de l'interface Terminal : il s'agit du fichier batch en entrée (1).

- il lance le Moniteur, qui travaillera comme auparavant.

- s'il existe un fichier résultat (4), jusqu'alors sous le contrôle de l'interface Fichiers Standard, il le transmet à l'utilisateur.

Si ce fichier n'existe pas, il envoie à l'utilisateur le fichier batch en sortie (2) (écran de l'utilisateur réseau) qui contient probablement un ou plusieurs messages d'erreurs, seuls "résultats" fournis par le Moniteur.

3. Le niveau IDML (langage "haut-niveau").

Le processus de travail est identique à celui du niveau Lgbs.
La seule différence, c'est qu'au lieu de recevoir un programme en Langage de base, le Moniteur Général reçoit un programme source et crée un fichier de commandes adéquat.

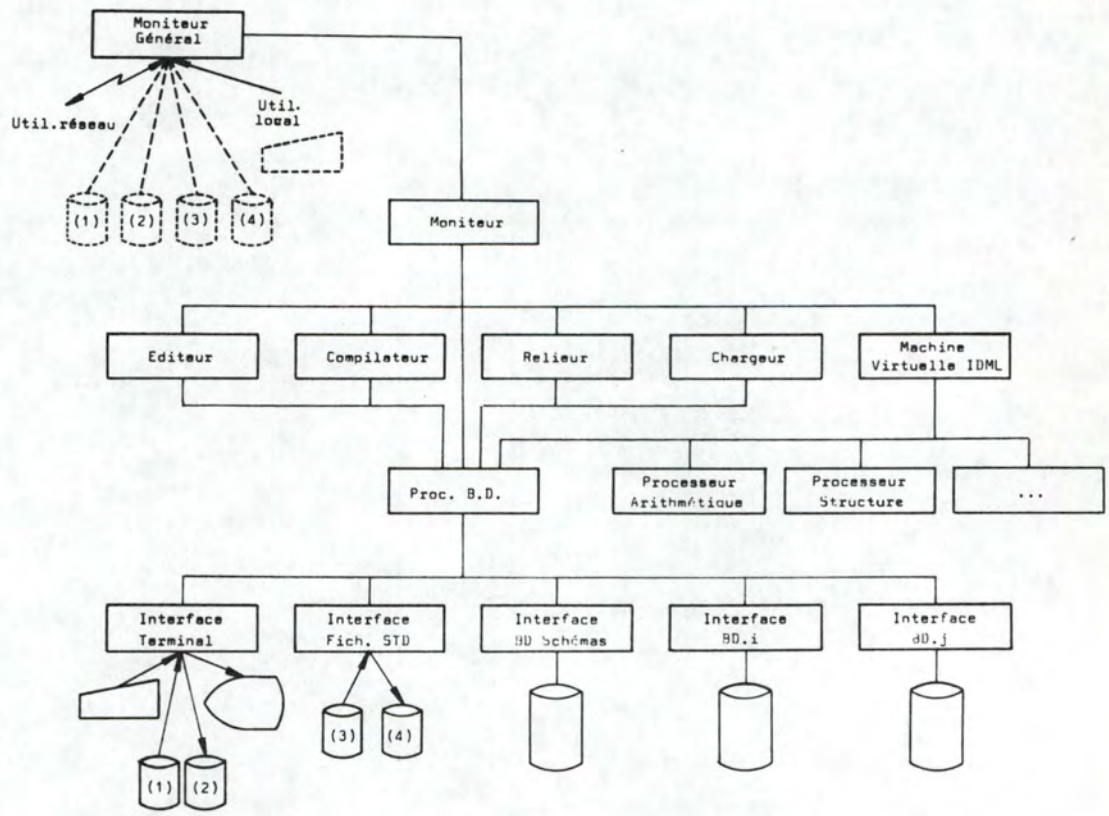


Fig. III.4. - Le système IDML partagé : niveau IDML.

3.1.4. Réalisation : l'environnement.

Le matériel dont on dispose est constitué de micro-ordinateurs North-Star Horizon.

Le North-Star est un micro-ordinateur piloté par un microprocesseur Z80A de ZILOG : il a une capacité mémoire (RAM) de 64 Kbytes et dispose d'une mémoire de stockage (Disquettes) de 179 Kbytes par unité (2 unités dans notre configuration; extension possible à 4 unités).Il possède, entre autre deux interfaces série RS 232 et une interface parallèle.

Une interface RS 232 est réservée à la connection d' un terminal; la seconde est dès lors disponible pour d'autres applications: dans ce cas, le réseau local.

Le langage de programmation dont nous disposions au départ sur le North-Star est le Pascal. Le Pascal North-Star est un système complet de développement de programmes.
Les programmes sources sont préparés à l'aide d'un éditeur orienté écran. Ils sont compilés dans un code interne, appelé P-code, et interprétés par le processeur Pascal, appelé P-machine.

Ce système Pascal est une version North-Star du système Pascal UCSD développé à l'université de Californie, San Diego.
Le Pascal North-Star s'exécute avec son propre système d'exploitation.

Mais la mise en oeuvre du réseau local en liaison avec le Pascal North-Star n'a pas pu être réalisée, des problèmes se posant au niveau des interruptions.

Il a donc été décidé de réaliser les programmes en Pascal MT+.

Le PASCAL MT+ a été développé par le groupe MT microSYSTEMS. Il supporte de nombreuses améliorations par rapport au Pascal Standard : dans les variables et les instructions du langage (types prédéfinis : le byte, le mot, le string; opération sur le string, option ELSE de l'instruction CASE); dans le système de gestion de fichiers (accès direct, E/S à grande vitesse); dans l'architecture du programme (compilation modulaire, structure d'overlays),... [voir annexe II].

Le Pascal MT+ est donc approprié pour l'exploitation d'applications de traitement de données.

Le Système Pascal MT+ comprend, entre autre, un compilateur, un relieur, un désassembleur, une librairie de sous-routines.
Il requiert une unité centrale Z80 et le système d'exploitation CPM.
Lors de l'exécution d'un programme, le système Pascal utilise 5 Kbytes d'espace mémoire. Dans une configuration 64 Kbytes, l'utilisateur a donc à sa disposition 59 K de mémoire centrale. Si le programme nécessite plus de place, il faut faire appel à la gestion d'overlays.

La réalisation a pris en compte la présence d'un utilisateur local et de deux utilisateurs-réseaux.

3.2. LE RESEAU

Introduction.

L'étude du réseau qui suit n'a pas de caractère général. Elle se place dans le cadre précis du réseau local qui a été réalisé et dont nous donnons une description (paragraphe 3.2.1.). La conception du système de transmission se base sur une architecture en couches.
Les couches inférieures réalisent des fonctions standard et procurent des services à la couche supérieure.

Nous décrirons tout d'abord les services que le programme d'application désire obtenir en vue d'une communication inter-processus (paragraphe 3.2.1.).

Ensuite, nous analyserons les fonctions de la couche application (paragraphe 3.2.2.) : il s'agit d'établir les principes du protocole de communication et de le réaliser.

La description du réseau local et la présentation des trois couches inférieures de l'architecture sont extraites du travail de E. Lemal (- Contribution à la mise en oeuvre d'un réseau local -).

3.2.1. Description du réseau local.

Le réseau local a pour but de relier entre eux des micro-ordinateurs North-Star Horizon. Chaque appareil est connecté au réseau par l'intermédiaire d'une interface RS 232.

La réalisation du réseau local a été faite sous le principe de l'architecture bus.

Les caractéristiques essentielles de ce réseau sont :

- nombre maximum de stations : 225;
- support de transmission utilisé : paire blindée de câbles torsadés;
- type de transmission : Standard RS 232 (Avis V24 du CCITT);
- taux de transfert : 4800 Bps;
- longueur maximum du réseau : à déterminer;
- procédure de contrôle des communications : basée sur un concept multi-accès sans "convention" avec détection et résolution de collision;

- longueur des messages échangés entre processus d'application : longueur variable, limitée à 250 octets.

Les objectifs recherchés dans la conception du réseau se résument aux propriétés suivantes :

simplicité, compatibilité avec d'autres systèmes, équité entre tous les noeuds, non immobilisation de toutes les ressources par un seul noeud, stabilité, fiabilité et maintenabilité, architecture en couches.

Un certain nombre d'options, telles la vitesse et la sécurité, ont été délibérément écartées.

L'architecture se décompose en quatre couches :
1. la couche application,
2. la couche transport,
3. la couche liaison de données,
4. la couche physique.

La couche application est directement liée aux programmes d'application, tandis que la réalisation des trois couches inférieures constitue le logiciel de base du réseau.

3.2.2. Les trois couches inférieures de l'architecture.

En partant des besoins globaux qu'ont les processus d'application, on aboutit finalement à la description des fonctions élémentaires nécessaires que doivent réaliser les trois couches inférieures de l'architecture (Transport, Liaison de données, Physique)

3.2.2.1. Les besoins globaux des processus d'application.

Pour l'instant, les besoins d'utilisation du réseau local se limitent à l'échange d'informations entre les différents processus d'application (PA) qui s'exécutent dans les stations connectées au réseau.

On entend par échange d'informations :

- l'envoi de messages à destination d'un autre PA;

- la réception de messages en provenance d'un autre PA;

Un message est une suite d'octets. Aucune restriction n'est faite quant au contenu de ces octets (données binaires, caractères de texte). La longueur d'un message est variable mais limitée à 250 octets.

Toute station est identifiée de manière unique par une adresse. Cette adresse lui permet notamment de reconnaître un message qui lui est destiné. L'adresse est constituée d'un nombre compris entre 1 et 255.
Puisqu'il n'y a jamais qu'un seul PA qui s'exécute à la fois dans chaque station, en adressant celle-ci on adresse aussi le PA qui s'y déroule.

3.2.2.2. Les services requis.

Les services requis par un PA, en vue de l'échange d'informations avec d'autres PA, doivent lui permettre :

- d'initialiser l'accès au réseau;
- de recevoir des messages en provenance de n'importe quelle station du réseau, i.e. de n'importe quel PA;
- d'envoyer des messages à n'importe quel PA du réseau;
- de clôturer l'accès au réseau lorsque le PA n'a plus besoin d'y accéder.

Ces quatre services sont fournis au PA sous forme d'appel à des procédures Pascal.
Voici l'énoncé de ces procédures (la spécification précise est disponible dans l'annexe III (- Implémentation du niveau Primitives -)).

1. OPEN_LINK (VAR host_add:address; VAR err:code)

   Demande d'initialisation d'une liaison virtuelle.

   Cette procédure permet au PA d'accéder au réseau. Elle renvoie au PA appelant l'adresse de la station (host_addr) et un code d'erreur éventuel (err), par exemple :

   - accès déjà ouvert;
   - autre station déjà connectée sous la même adresse.

2. CLOSE_LINK (VAR err:code)

   Demande de clôture.

   Cette procédure a pour but de fermer l'accès au réseau. Après l'appel à cette procédure, la station est déconnectée du réseau, de sorte que le PA ne puisse plus recevoir de messages en provenance des autres PA qui accèdent au réseau. Un code d'erreur (err) est retourné au PA, indiquant les conditions de réalisation de cette procédure :

   - accès déjà fermé;
   - messages reçus mais non retirés par le PA.

3. SEND (to_addr:address; len:length; msg:message; VAR err:code)

Demande d'envoi d'un message.

Cette procédure a pour but d'envoyer un message (msg) dont on connait la longueur (len) à la station dont on spécifie l'adresse (to_addr).
Pour des raisons diverses, il se peut que le destinataire n'ait pas reçu le message ou l'ait refusé; la procédure d'envoi applique la politique du meilleur effort pour assurer un service sûr et fiable au PA utilisateur.
Le PA peut savoir dans quelles conditions l'envoi s'est réalisé; dans ce but, la procédure renvoie au PA un code d'erreur (err), par exemple :

- trop d'essais;
- station destinatrice inexistante.

4. RECEIVE (VAR from_addr:addresse; VAR len:length; VAR msg:message; wait:flag, VAR err:code).

Demande de réception d'un message.

Deux solutions sont possibles lors de l'appel à cette procédure. Dans la première solution (wait=vrai), un message peut être déjà disponible, auquel cas la procédure renvoie ce message (msg), sa longueur (len) et l'adresse de son expéditeur (from_addr).
Si aucun message n'est disponible lors de l'appel à cette procédure, le PA est bloqué jusqu'à disponibilité d'un nouveau message.
Dans la seconde solution (wait=faux), si aucun message n'est disponible, le PA n'est pas bloqué et un code d'erreur (err) est renvoyé.
Si un message est disponible, il est fourni au PA comme dans le cas précédent.

N.B. : Un PA peut faire appel à cette procédure, même après clôture de l'accès au réseau, pour prendre les messages restant qui ont été reçus avant cette clôture.

3.2.2.3. Synchronisme et asynchronisme des services.

Certaines exigences sont faites quant à la fiabilité de réalisation de ces services.

On souhaite qu'un message soit correctement reçu par le destinataire. Si l'envoi n'a pu s'effectuer dans les conditions espérées (à la suite de plusieurs essais, le message n'a toujours pas été correctement reçu ou n'a pas été du tout reçu, à cause de mauvaises conditions de transmissions, parce que la station destinatrice est absente, ou parce que le réseau est fort chargé et que le message est entré en collision avec d'autres messages à

plusieurs reprises), on désire que le PA en soit averti pour qu'il puisse prendre des mesures appropriées.

Ces considérations imposent à la procédure d'envoi (SEND) qu'elle se déroule de façon synchrone par rapport au PA puisqu'il doit, en effet, savoir dans quelles conditions le service d'envoi s'est terminé.

Cette fiabilité, au niveau de l'expédition de messages, exige des stations connectées au réseau, qu'elles puissent, en permanence, recevoir les messages qui leur sont destinés. Il est souhaitable, dès lors, que ces messages reçus soient aussitôt stockés et que le PA émetteur soit averti de leur bonne réception.

Le stockage des messages reçus se fait dans un tampon - appelé panier - prévu à cet effet, jusqu'à concurrence de l'espace disponible.
Les messages reçus et stockés dans le panier peuvent ensuite être délivrés au PA au fur et à mesure que celui-ci en fait la demande (RECEIVE), jusqu'à concurrence du nombre de messages disponibles.

Le mécanisme de réception et de stockage que l'on vient d'expliquer, et par lequel des messages peuvent être reçus et stockés sans l'intervention du PA, se déroule de façon asynchrone par rapport à l'exécution de ce PA.

3.2.3. La couche application.

3.2.3.1. Les principes du protocole de communication.

Le protocole de communication que nous allons décrire est donc un protocole entre programmes d'application connus, au niveau des trois couches supérieures (découpe ISO).
Trois caractéristiques nous ont paru essentielles dans la conception de ces règles de dialogue.

Tout d'abord, il faut que ce protocole assure un maximum de sécurité pour les PA : il faut éviter qu'un PA ne cesse de travailler parce qu'il est bloqué sur un problème réseau. Toute difficulté du réseau doit donc pouvoir être découverte et distinguée des autres, afin de pouvoir réagir en conséquence.

La perception de la plupart de ces difficultés est prise en charge par les couches inférieures du système de communication. Dès lors, une grande attention est portée à la valeur du code d'erreur fournie par les primitives.

Cependant, certains phénomènes perçus par un site peuvent concerner non pas le réseau mais le site correspondant. Il faut donc que celui-ci avertisse l'autre site de son état.

Par exemple, si le site de traitement ne peut prendre en compte une demande, il faut qu'il le fasse savoir au site demandeur. Nous ne sommes donc plus devant un problème réseau mais bien un problème lié directement aux sites.

Ensuite, il faut que cette sécurité soit assurée avec un minimum de moyens, c'est-à-dire avec un faible trafic sur le réseau. Toute lourdeur dûe aux messages de protocole doit être évitée. Nous avons là un objectif de performance-trafic.

Enfin, il s'agit de ne pas oublier que les utilisateurs sont nombreux. Dès lors, si chacun se met à envoyer de multiples messages, le site-récepteur va être rapidement saturé. C'est pourquoi nous devons veiller à ce que, sauf dans un cas précis que nous signalerons par la suite, il n'y ait maximum qu'un seul message de chaque utilisateur potentiel dans le panier de réception.
Nous avons à faire face à un problème de performance-disponibilité attente.

Deux protocoles ont été alors conçus :
le premier concerne la communication au niveau Primitive,
le deuxième concerne le travail au niveau du Langage de base ou du Langage IDML.

Cette distinction a été faite afin, comme nous l'analyserons ensuite, que chaque protocole respecte au mieux les caractéristiques reprises ci-avant.

3.2.3.2. Réalisation du protocole - niveau Primitive.

1. Les différents types de messages.

   - les messages de protocole :

     P1 demande de communication au niveau primitive
     P2 acknowledgement
     P3 fin de communication.

   - les messages de données :

     D1 requête-primitive
     D2 réponse-primitive.

2. L'amplitude du protocole.

   Par amplitude, nous voulons désigner la dimension fonctionnelle du protocole : quel travail couvre-t-il ?

   Deux possibilités étaient présentes :

   - d'une part, choisir la primitive comme unité de travail,

   - d'autre part, définir la session-utilisateur comme unité de base.

   Dans le premier cas, cela signifie que chaque message requête-primitive (D1) ou réponse- primitive (D2) doit être entouré de messages de protocole adéquats (communication, acknowledgement...), d'où une évidente lourdeur.

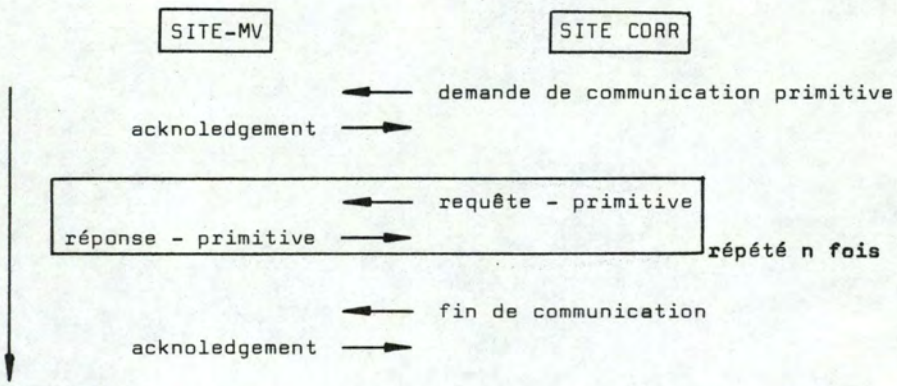
   Dans le second cas, nous avons dès lors envisagé de diminuer ce nombre de messages en considérant que l'ensemble des requêtes-primitives et réponses-primitives doit être protégé par des messages particuliers.
   Ainsi nous réalisons l'objectif de performance-trafic.

   Reste la question de sécurité.
   Vu que les primitives de base du réseau (émission d'un message) renseignent l'utilisateur quant à la terminaison du service demandé, et que, d'autre part, le temps écoulé entre l'envoi d'une requête-primitive et la disponibilité de la réponse-primitive est limité, nous n'avons pas jugé nécessaire de répondre à une requête-primitive par un message d'acknowledgement, la réponse-primitive jouant ce rôle.

   Nous avons donc retenu la session-utilisateur comme unité de travail.

3. Exemple de déroulement du protocole dans les conditions optimales.

Fig. III.5.



4. Structure générale des échanges pour l'exécution d'une session-utilisateur.

Le formalisme utilisé pour les arbres programmatiques [AP] est issu de la théorie de Jackson. Il est explicité dans l'annexe I.

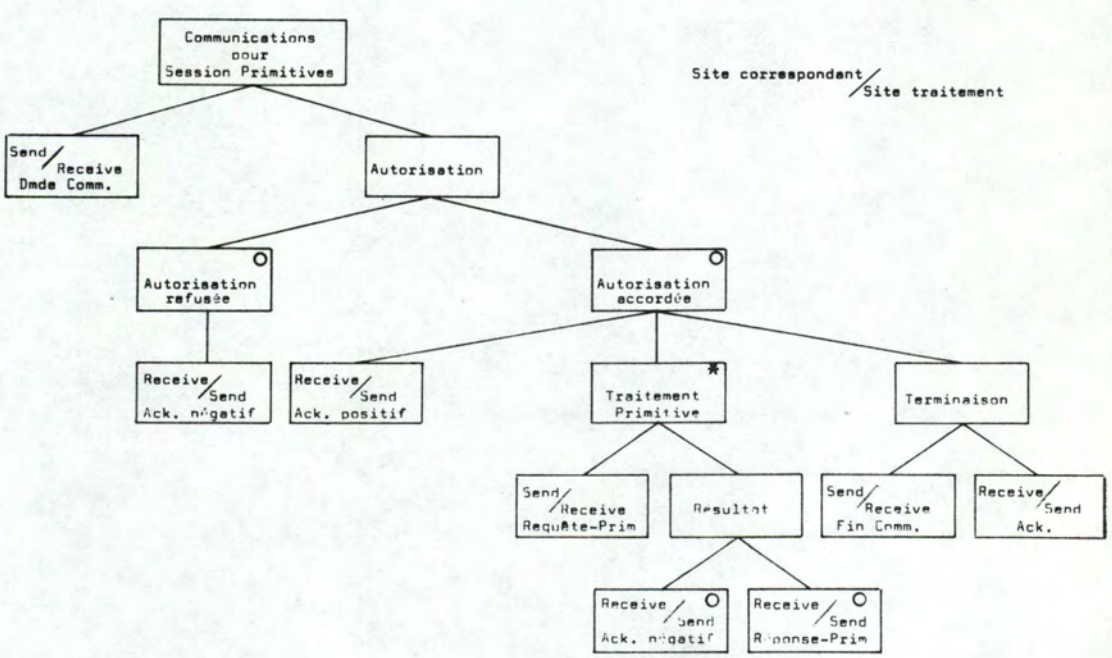


Fig. III.6. - [AP] Le protocole de niveau Primitives.

3.2.3.3. Réalisation du protocole au niveau Langage de base et au niveau Langage IDML.

1. Les différents types de messages.

   - les messages de protocole :

     P1 demande de communication niveau II, III
     P2 acknowledgement
     P3 fin de communication.

   - les messages de données :

     D1 bloc de caractères du programme en Langage de base
     D2 bloc de caractères du programme en Langage IDML
     D3 bloc de caractères résultat.

2. L'amplitude du protocole.

   A nouveau, deux possibilités se présentaient à nous :

   - d'une part, considérer que l'unité de communication est un programme en langage de base, un programme en langage IDML ou un ensemble de résultats,

   - d'autre part, se baser sur la session-utilisateur (c'est-à-dire programme en langage de base ou programme en langage IDML et ensemble de résultats).

   Vu que le traitement interne entre la réception d'un programme et l'émission des résultats peut être assez long (exécution d'un programme et éventuellement compilation), il est vite apparu indispensable de considérer ces deux évènements séparés, comme deux communications distinctes.

   Cela laisse plus d'indépendance aux programmes d'application. Le trafic est certes un peu plus dense sur le réseau, mais les quelques messages supplémentaires représentent une charge faible par rapport au contenu des communications proprement dites.

   Nous avons donc choisi comme unité de travail
   - ou le programme en langage de base,
   - ou le programme en langage IDML,
   - ou les résultats d'une exécution.

Remarque : nous référant au chapitre 3.1.3. (- Les trois architectures -), nous pouvons voir que cette notion d'unité de travail correspond au transfert d'un fichier.

3. Exemples du déroulement du protocole, dans les conditions optimales.

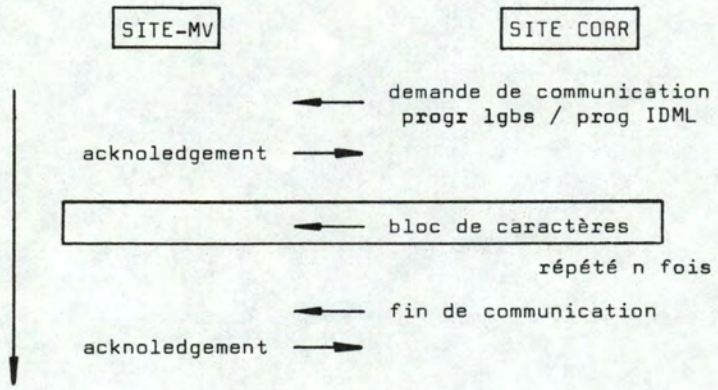


Fig. III.7. - Transfert d'un programme.

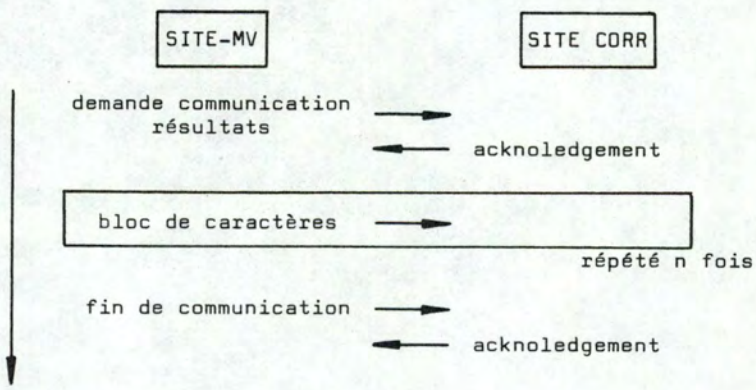


Fig. III.8. - Transfert de résultats.

4. Structure générale des échanges pour l'exécution d'une session- utilisateur.

Voir la Figure III.9. (- page suivante -).

3.2.3.4. Respect des objectifs.

Lors de la présentation de chaque protocole, nous avons essayé de montrer comment il rencontrait l'objectif de sécurité et l'objectif de performance-trafic.

Il nous reste à soulever la question de performance-place mémoire.

Cet objectif est rencontrê vu que le dialogue est une succession de messages, le premier allant de A vers B, le suivant de B vers A, puis A vers B...
Un utilisateur n'enverra donc pas deux messages consêcutifs à un même site.

Sauf dans un cas : lors de l'envoi de bloc-programme ou bloc-rêsultat. Cependant, vu que nous sommes certains que le site rêcepteur est prêt (puisqu'il a signalê sa disponibilitê avec le message d'acknowledgement), les messages êmis successivement seront aussitôt rêceptionnês. Nous êviterons ainsi une surcharge de la file d'attente, celle-ci pouvant être utile comme tampon entre l'êmetteur et le rêcepteur, quand le traitement à la rêception est lègèrement plus long.
Mais nous n'assisterons qu'à un phênomène limitê dans le temps, qui ne dêfavorisera aucun autre utilisateur puisque le programme d'application ne saurait de toute manière pas traiter, à ce moment, cet autre utilisateur potentiel.

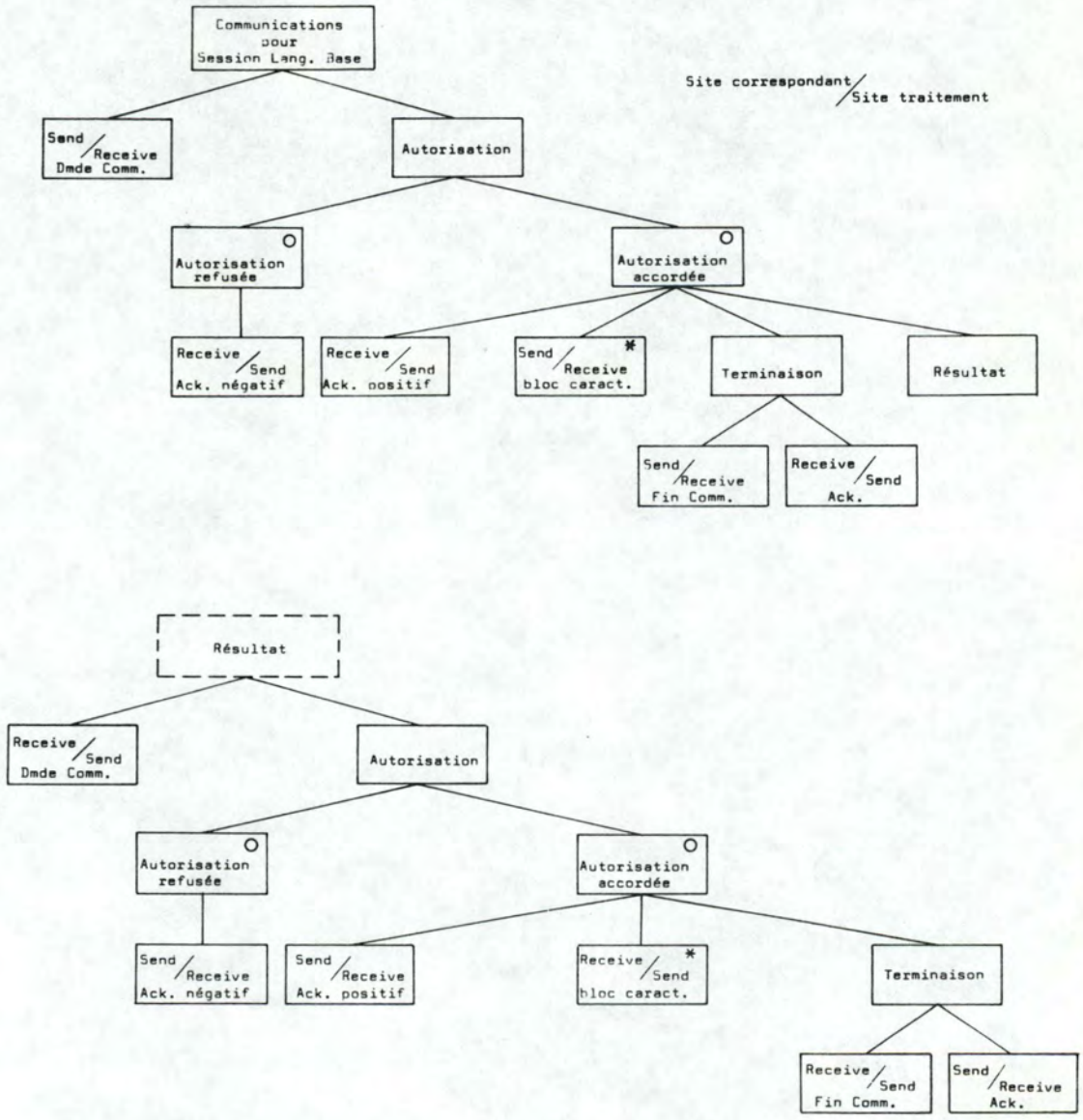


Fig. III.9. - [AP] Le protocole de niveau Lgbs/Idml.

3.3. IMPLEMENTATION DU NIVEAU - PRIMITIVES - .

3.3.1. Rappel de l'architecture.

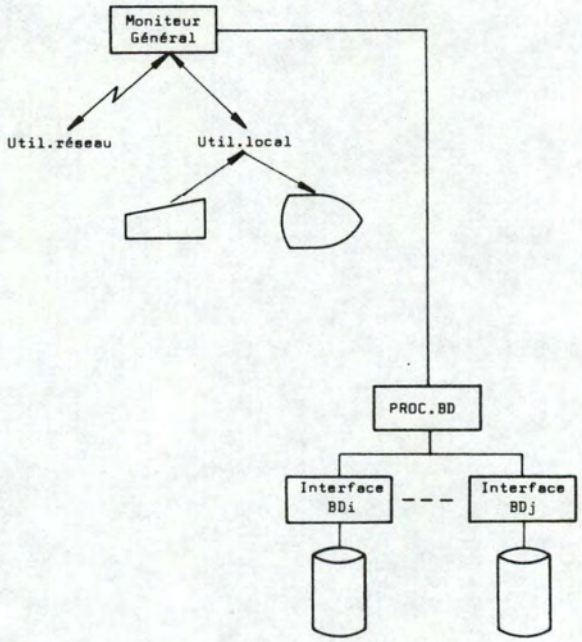


Fig. III.10. - Le système IDML partagé : niveau Primitives.

3.3.2. Le rôle du Moniteur Général.

Le Moniteur Général doit remplir deux fonctions :

- interface entre un utilisateur (local ou réseau) et le système IDML,
- moniteur d'un système multi-utilisateur en temps réel.

3.3.2.1. Interface entre l'utilisateur et le système IDML.

Le travail du Moniteur Général diffère en fonction du type d'utilisateur :

- utilisateur-local
- utilisateur-réseau.

L'utilisateur-local.

La procédure de travail de l'utilisateur local qui aboutit à la demande d'exécution de primitives peut être vue de trois façons différentes.

- le travail est totalement interactif,
- les primitives sont extraites d'un fichier,
- les primitives sont générées par l'exécution d'un programme dans un langage classique.

Nous n'avons pas tenu compte de ces deux dernières possibilités.
La seule procédure implémentée et évaluée est l'émission et la réception interactive des primitives.

Rappelons en outre que l'utilisateur local est unique.

Le schéma d'exécution se déroule en trois parties :

1. collecte des données pour constituer la requête (primitive),

2. lancement de l'exécution : appel du système IDML (en particulier le processeur Base de Données),

3. visualisation du résultat de cette exécution.

1. Constitution de la requête-primitive.

Comme nous l'avons vu au chapitre 2.5. (- Les interfaces Base de Données -), une requête-primitive consiste en un ensemble de paramètres. Certains de ces arguments doivent avoir une valeur, afin de constituer une primitive.
La collecte de ces valeurs se fait de façon interactive (clavier et écran) avec l'utilisateur.
Le Moniteur Général le guide dans les paramètres multiples d'après la valeur qu'il a donnée au paramètre COP (code operation). De plus, afin de faciliter le travail de l'utilisateur, le Moniteur Général rappelle pour chaque paramètre la dernière valeur que l'utilisateur lui a attribuée.

2. Appel du système IDML.

Le Moniteur Général passe le contrôle au processeur Base de Données et lui fournit l'adresse des paramètres de la requête.

3. Visualisation de la réponse-primitive.

Une fois l'exécution terminée, le processeur Base de Données rend la main au Moniteur Général et lui restitue les paramètres dûment modifiés, qui constituent la réponse. Le Moniteur Général affiche sur l'écran ces arguments modifiés et laisse, éventuellement, une trace de cette exécution dans un fichier résultat.

L'utilisateur-réseau.

Le schéma d'exécution repose sur les mêmes principes :

1. réception d'une requête-primitive,

2. appel du système IDML,

3. émission de la réponse primitive.

Un message qui transite sur le réseau est une suite d'octets. Si le contenu de ces octets est quelconque vis-à-vis du réseau, il n'en est évidemment pas de même pour le programme d'application.

Le message-primitive est composé de trois parties :

- l'en-tête qui comprend le type du message et une indication de chaînage,

- les adresses relatives des paramètres,

- la partie données proprement dite.

Afin d'éviter un trop lourd trafic, il faut véhiculer le moins possible d'octets inutiles. Les paramètres ayant des tailles standard, nous pouvons affirmer que deux messages-réseau sont suffisants pour transmettre une primitive.

Le deuxième message n'étant pas toujours nécessaire, une indication de chaînage a été introduite dans l'en-tête du premier message-réseau.

1. Réception d'une requête-primitive.

   Le rôle du Moniteur-Général consiste à réceptionner le message et à en vérifier le type.
   Une fois ces opérations réalisées positivement, le Moniteur Général calculera les adresses réelles des paramètres (sur base des adresses relatives contenues dans le message).

2. Appel du système IDML.

   Le moniteur cède le contrôle au processeur Base de Données. Il lui donne l'adresse des différents paramètres ainsi qu'une référence interne de l'utilisateur.
   Cette référence est nécessaire pour que les interfaces puissent convenablement gérer les Bases de Données-multi utilisateurs.

3. Emission de la réponse-primitive.

   L'exécution terminée, le Moniteur Général reprend l'initiative. Il lui reste à envoyer au site émetteur les paramètres qui constituent la réponse-primitive.

La structure du message de réponse est identique à celle du message de requête. La longueur effective de la réponse peut être calculée sur base des adresses relatives.

3.3.2.2. Moniteur d'un système "multi-utilisateurs en temps réel".

Quelle est la situation ?

Nous avons un site avec une unité centrale, mono-programmable. Sur ce site doit se dérouler un programme d'application (le système IDML) commun à de nombreux utilisateurs.
Chaque utilisateur ignore l'existence des autres et désire exploiter le système IDML dans les meilleures conditions.

Que doit-on réaliser ?

Nous voulons que le programme d'application tende à une exécution en temps réel pour chaque utilisateur.

Définition du système.

Rappelons tout d'abord quelques concepts :

- Temps réel (Real Time) :

  utilisation de l'ordinateur dans laquelle ce dernier doit élaborer, à partir d'informations acquises d'un processus extérieur, des informations de commande ou de contrôle dans un temps cohérent avec l'évolution du processus concerné.

- Temps partagé (Time Sharing) :

  utilisation simultanée d'un même ordinateur à partir de multiples terminaux. Une tranche de temps (Time Slice) est en général accordée successivement à chaque utilisateur.

- Multi-programmation :

  exécution imbriquée dans le temps de plusieurs programmes par un même ordinateur.

Le système que nous voulons construire ne peut pas être considéré strictement comme un système à temps partagé ou un système de multi-programmation.

Néanmoins, les principes sont similaires. Et nous pouvons définir ainsi le système multi-utilisateur "temps réel" :

> Exécution d'un programme utilisé simultanément par plusieurs utilisateurs, de façon indépendante. Une tranche de temps (Time Slice) est accordée successivement à chaque utilisateur.

L'utilisateur a ainsi l'illusion que le programme d'application est exécuté pour lui seul, et ce en temps réel.

Principes de réalisation.

Il n'est pas nécessaire d'avoir un support multi-tâches pour pouvoir programmer des systèmes qui demandent à gérer plusieurs processus simultanément .
Le programmeur doit écrire un logiciel propre, où il gère l'ordonnancement de ces processus de façon à maintenir un temps de réponse adéquat pour tous les utilisateurs, où il contrôle la réentrance et l'accès aux variables communes.

Il est donc impossible d'exécuter rigoureusement en parallèle plusieurs processus. On va les exécuter quasi parallèlement, c'est-à-dire qu'on va exécuter une partie du processus du premier utilisateur, puis une partie du processus d'un second utilisateur, et ainsi de suite.
Il faut alors continuer le processus du premier utilisateur à l'endroit où il a été interrompu.
A chaque changement de processus, il est dès lors nécessaire de sauver l'état de l'utilisateur qui est interrompu et de restaurer l'état de l'utilisateur qui va être traité. C'est la commutation de contexte.

Une des méthodes utilisées pour déterminer à quel moment il faut passer d'un processus à un autre, est celle de la tranche de temps. Une horloge externe force le processeur à se débrancher du processus en cours et à entrer dans une routine qui fera la commutation des processus.

Mais cette solution n'a pu être réalisée : le programme Pascal ne peut pas accéder à une horloge temps réel. Il a donc fallu trouver une solution de remplacement pouvant réaliser les mêmes fonctions.
Le principe général est le suivant : le procesus est subdivisé en étapes. Chaque fois qu'on entre dans le processus, une seule étape est exécutée, puis le système passe à un autre processus.

Reste à définir ces étapes, définition qui est liée au degré de parallélisme souhaité.

Examinons deux cas :

1. Si 10 utilisateurs ont au même moment chacun un programme à exécuter, et ce pour une heure, un parallélisme élevé est mauvais : en effet, ce n'est qu'après 10 heures que les 10 utilisateurs auront terminé.
   Si on avait traité successivement chaque utilisateur, le 10ème utilisateur ne serait ni avancé ni désavantagé, mais les 9 autres auraient gagné entre 1 et 9 heures.

2. Si un utilisateur a une exécution du programme d'une durée de 10 heures et 9 autres un programme d'une minute, un parallélisme élevé est avantageux.

En effet, quel que soit l'ordre dans lequel les programmes sont exécutés, les 9 programmes rapides seront terminés après 10 minutes au maximum et le lent après 1 h.10 minutes. Dans un traitement séquentiel, les performances seraient fonction de l'ordonnancement.

Ces deux exemples montrent qu'afin d'optimiser les temps de passage, il faudrait prévoir des priorités en fonction de la durée.
Malheureusement, dans le problème qui nous intéresse, on ne peut pas déterminer a priori qui est l'utilisateur le plus important.

Nous avons donc pris l'option de mettre tous les utilisateurs au même rang de priorité, et de subdiviser le processus en étapes logiques en prenant garde à rester le moins longtemps possible inactif, cette inaction étant due à l'attente de périphériques.

Cette attente a pu être évitée en particulier grâce aux mécanismes offerts par le réseau : un message venant du réseau interrompt momentanément le programme d'application. Une routine se charge alors de réceptionner le message et de le placer dans un panier de réception, puis rend la main au programme d'application. Le processus d'application n'a alors plus qu'à le sortir de ce panier pour pouvoir le traiter.

3.3.3. Le logiciel réalisé.

3.3.3.1. Architecture du programme du site de traitement.

L'architecture est calquée sur celle du système IDML. Chaque module du système a un équivalent en terme de module de programmation.

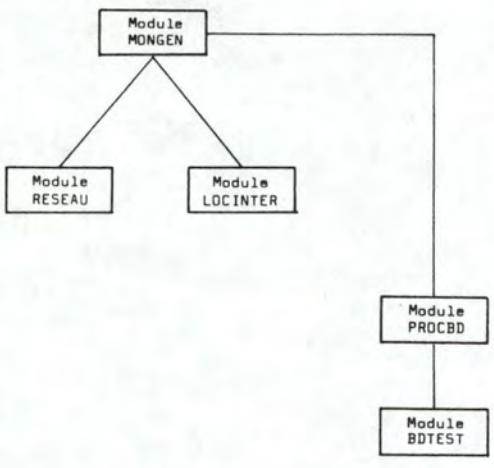


Fig. III.11. - Réalisation du niveau Primitives.

Au sens Pascal MT+, un module est un ensemble de variables et de procédures qui forment une unité de compilation.
Les procédures peuvent faire appel à des variables ou procédures déclarées dans d'autres modules.

Cette structure permet une architecture très modulaire.

Nous allons pour chacun des modules citer les fonctions principales qu'ils recouvrent. Ces fonctions mettent en pratique les principes énoncés dans ce chapitre.

Pour plus de détails sur les procédures, se référer en annexe III.

1. Le module Moniteur Général MONGEN.

En tant que moniteur du "système multi-utilisateurs en temps réel", le Moniteur Général se doit d'allouer l'unité centrale successivement à chaque utilisateur, et ce pour la durée d'une étape.

Ce que nous avons appelé processus correspond à une session-utilisateur et en particulier ici l'exécution d'un ensemble de primitives. Chaque étape du processus est constituée d'une action logique dans cette session.

Déroulement du système multi-utilisateurs.

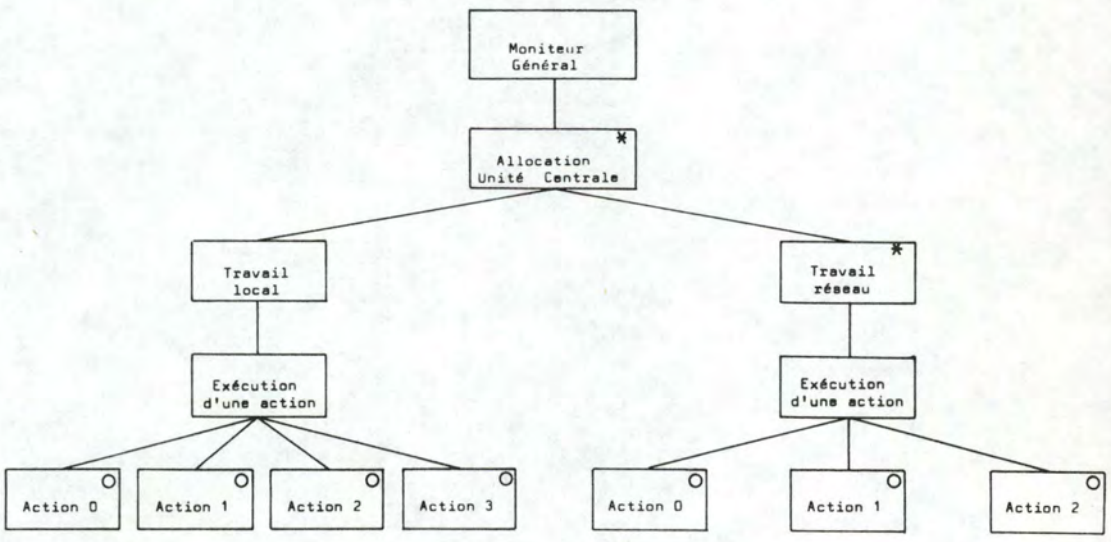


Fig. III.12. - [AP] Le programme SIDMLS.

Découpe en actions du travail de l'utilisateur local.

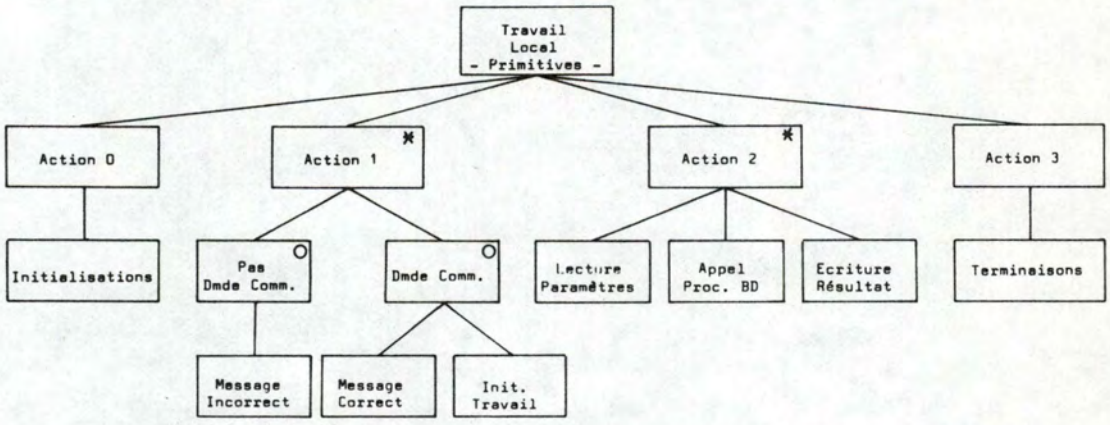


Fig. III.13. - [AP] La procédure de Travail Local.

Découpe en actions du travail d'un utilisateur réseau.

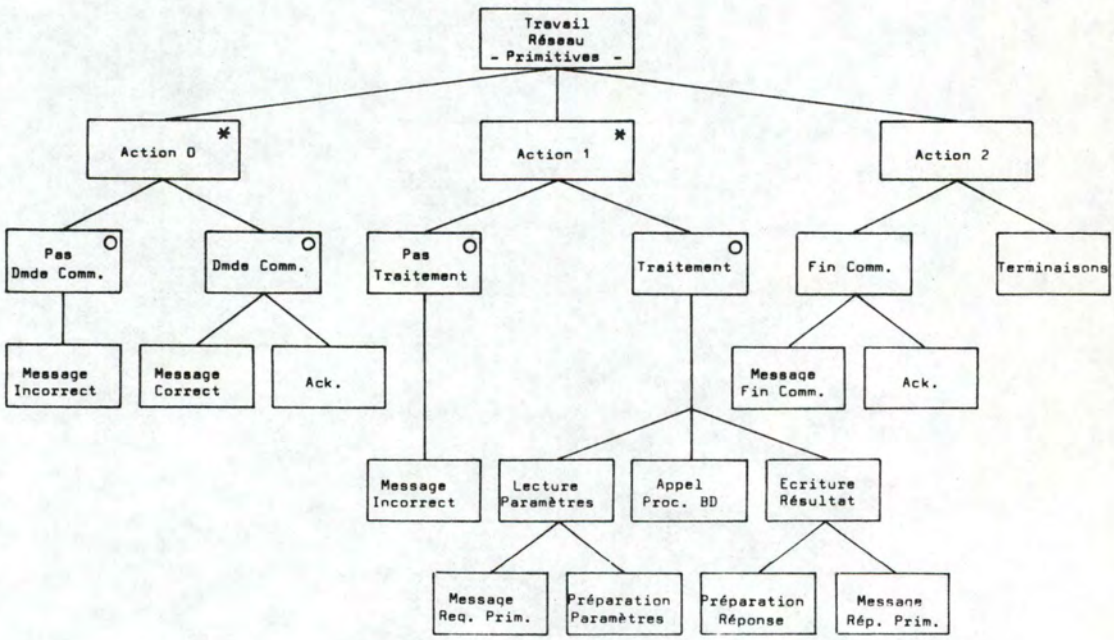


Fig. III.14. - [AP] La procédure de Travail Réseau.

2. Le module réseau.

Il met en oeuvre tous les éléments directement liés à un utilisateur réseau, dans le cadre d'un travail où il s'exprime par primitives.

Les procédures sont de deux niveaux :

- application : il s'agit de fournir dans une zone de travail, les paramètres de la primitive à traiter (réception d'une requête primitive RD-NET-PRIM) ou d'envoyer les résultats de l'exécution de cette primitive dont les valeurs sont dans cette même zone de travail (émission d'une réponse primitive WR-NET-PRIM). Une primitive nécessite 1 à 2 messages sur le réseau.

- liaison avec le réseau : les procédures de réception (NET-READ) et d'émission (NET-WRITE) d'un message ont pour but de lire ou écrire un bloc de caractères sur le réseau et de le mettre à la disposition du programme. Elles font appel aux procédures du logiciel de base du réseau et elles prennent en charge les traitements à effectuer en cas d'erreur ou de problème.

  En outre, la procédure de lecture peut vider le panier de réception des messages arrivés mais que le système ne sait pas traiter au moment donné. Ces messages en attente sont dès lors stockés dans des tampons prévus à cet effet.

  Cette gestion a été nécessaire vu que la lecture du panier de réception se fait selon le principe FIFO (First In, First Out). Il est donc apparu nécessaire de pouvoir vider cette file quand on désire obtenir le message d'un utilisateur particulier.

3. Le module Locinter.

Ce module prend en charge le travail interactif de l'utilisateur local.

Le travail principal est de collecter les valeurs des paramètres qui constituent la primitive à exécuter, et d'afficher le résultat de cette exécution sur l'écran (et éventuellement dans un fichier).

Il s'agit dès lors d'un petit gestionnaire d'écran, conçu dans le cadre précis des primitives.
C'est ainsi qu'une fois le code opération donné, l'utilisateur n'a qu'à suivre les indications qui lui sont données afin de remplir les paramètres adéquats.

En outre, les valeurs attribuées lors de l'appel précédent sont restituées à l'utilisateur qui peut les garder telles qu'elles sont ou les modifier.

Le caractère rébarbatif lié aux codes utilisés pour citer les arguments est quelque peu diminué par l'existence de messages d'aide que l'utilisateur peut appeler à volonté. Attention, il s'agit uniquement de clarifier des codes

mnémoniques et de rappeler les différentes valeurs possibles d'un argument.
Aucun contrôle de validité n'est effectué par ce gestionnaire. Ce qui lui permet d'être utilisé quelle que l'interface à laquelle s'adresse la primitive.

Les résultats sont fournis à l'écran et ils peuvent faire l'objet d'une écriture dans un fichier (tous les paramètres ne seront pas repris, il s'agit de réaliser la trace des opérations).

Les procédures principales consistent à lire une requête primitive (RD-REQ-PRIM) et à écrire une réponse primitive (WR-REQ-PRIM).

La lecture consiste à lire le code-opération, afficher la valeur courante des arguments liés à ce code-opération (WR-ACTUAL-VALUE) et à s'informer des nouvelles valeurs (RD-NEW-VALUE).

L'écriture consiste à afficher, selon le code-opération, certaines valeurs d'arguments (WR-ACTUAL-VALUE) sur l'écran (WR-TERM-RES) et éventuellement dans un fichier (WR-FILE-RES).

4. Le module PROC-BD.

Recevant l'adresse des paramètres d'une primitive et la référence de l'utilisateur, le Proc BD répercute la requête vers l'interface concernée.

Les interfaces sous son contrôle sont celles qui gèrent une Base de Données conventionnelle multi-utilisateurs.

L'adjonction d'une Base de Données supplémentaire et de son interface nécessite la mise à jour du processeur de Bases de Données.

5. L'interface BDTEST.

Dans notre réalisation, cette Base de Données est unique. Nous l'avons nommée BDTEST. Elle est sous le contrôle d'une interface, qui assure la liaison entre le système IDML et le Système de Gestion de Fichiers Pascal MT+ [annexe II].

L'interface est décrite en détail en annexe. Elle met en oeuvre des concepts du modèle d'accès appliqués au SGF Pascal MT+.

3.3.3.2. Difficultés de la réalisation.

Deux grands problèmes ont surgi pendant la réalisation du niveau "primitives" :
le premier est lié à la taille du programme,
le second a trait au réseau.

Dans notre configuration, le North-Star a une zone mémoire de 64 Kbytes dont 59 sont disponibles pour le programmeur. C'est évidemment peu pour établir le système IDML.
D'une part le code et d'autre part les données ont besoin d'un espace nettement plus important.

Afin de surmonter cet obstacle, le Système Pascal MT+ offre deux outils : la structure des modules et le gestion d'overlays.

Cette technique de recouvrement permet de tirer un parti maximum de la mémoire, en n'y chargeant, à un instant donné, que le code nécessaire.

La gestion laisse beaucoup d'initiative au programmeur. A lui de traiter toutes les adresses réelles d'implémentation en mémoire.

Les conséquences de l'emploi de cette technique sont nombreuses :

- la mise au point est très lente : chaque byte utilisé a son importance; les opérations de reliage sont nombreuses et fastidieuses;

- le comportement du programme est parfois inattendu, ce qui oblige à changer la structure et perdre certains avantages de la modularité;
  par exemple, il ne faut pas déclarer un fichier dans un module, car le comportement du SGF est imprévisible. Il est donc absolument nécessaire de les déclarer tous dans le programme principal !

- les performances lors de l'exploitation sont touchées chaque fois qu'il faut charger en mémoire un autre module, ce qui nécessite des accès disques.
  A nouveau, il faut repenser la structure du programme, afin de minimiser ces chargements. La structure logique se voit encore mise en danger.

Le deuxième problème est lié au réseau, et en particulier au phénomène d'interruption. Tout d'abord, la nécessité de pouvoir interrompre un programme en cours, nous a forcé à passer du Pascal UCSD au Pascal MT+.
En effet, le Pascal UCSD supporte difficilement une gestion d'interruption du fait qu'il est compilé dans un code interne qui est ensuite interprété.
En outre, la liaison avec des procédures écrites en Assembleur est difficile.

Mais cette question n'est pas la plus importante, bien qu'elle ait ralenti assez fort la mise au point.

Ce qu'il faut regarder, ce sont les performances atteintes ou non par le réseau. Deux éléments du système mettent ces résultats en difficultés :

- pendant un accès disque, l'interruption n'est pas prise en compte;
  or, nous avons vu que ces accès sont nombreux, d'une part par la technique d'overlay et d'autre part par le fait que

nous travaillons sur une Base de Données et donc sur des fichiers en mémoire secondaire;

- les opérations de lecture et écriture au terminal sont difficilement interruptibles; or, quand l'utilisateur local travaille de façon interactive, ces opérations sont nombreuses. Les utilisateurs réseaux sont alors défavorisés, vu la difficulté qu'ont leurs messages d'être réceptionnés par le site de traitement.

En d'autres mots, de trop nombreuses tentatives de transfert doivent être réalisées avant qu'un message ne parvienne à destination. Ce qui dégrade le service vis-à-vis des utilisateurs-réseau.

3.3.3.3. Architecture du programme du site utilisateur réseau.

Elle est assez simple et met en oeuvre des modules déjà rencontrés :

- le module Réseau qui transmet la requête et réceptionne la réponse,

- le module Locinter qui permet de constituer la requête et de visualiser la réponse.

3.4. IMPLEMENTATION DU NIVEAU - LANGAGE DE BASE - .

3.4.1. Rappel de l'architecture.

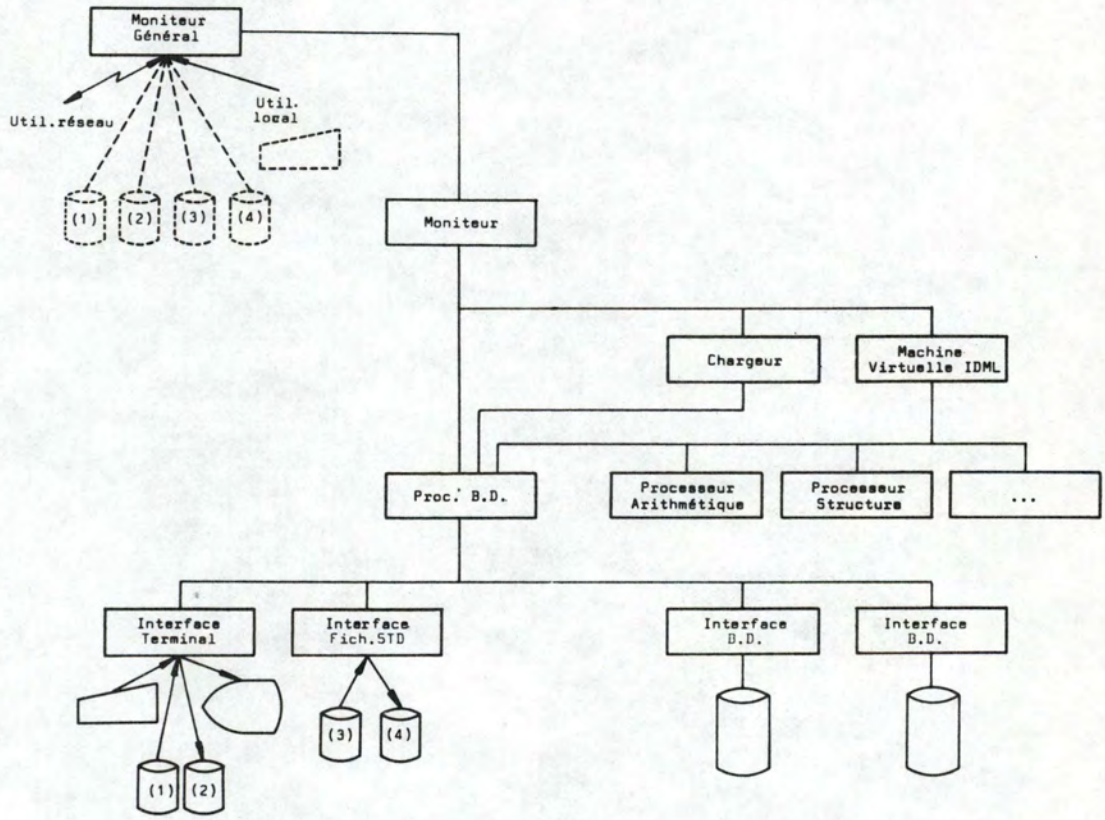


Fig. III.15. - Le système IDML partagé : niveau Lgbs.

3.4.2. Le rôle du Moniteur Général.

Le Moniteur Général a les deux mêmes fonctions que précédemment.

- interface entre un utilisateur (local ou réseau) et le système IDML,
- moniteur d'un système multi-utilisateurs en temps réel.

3.4.2.1. Interface entre l'utilisateur et le système IDML.

Le travail du Moniteur Général diffère en fonction du type d'utilisateur :

- utilisateur-local
- utilisateur-réseau.

L'utilisateur local :

Le rôle du Moniteur Général est très limité : l'utilisateur local ayant informé le Moniteur Général qu'il veut travailler avec le système IDML au niveau du langage de base, le Moniteur Général le met en relation directe avec le Moniteur du Système IDML, qui a dorénavant l'initiative, et ce jusqu'à la fin de la session.

Le Moniteur peut recevoir les commandes de façon interactive ou non. Les ordres d'opération doivent alors se trouver dans le fichier batch.

L'utilisateur réseau :

Il ne suffit plus ici de passer directement le contrôle au Moniteur du Système IDML, et ce pour plusieurs raisons.

Tout d'abord, le système IDML établi un dialogue avec l'utilisateur.
Ce dialogue devrait alors se faire par l'intermédiaire du réseau, ce qui est fort lourd.

Ensuite, le fichier de travail (essentiellement le fichier contenant le programme exécutable) n'est pas disponible sur le site de traitement. Il faut donc le transférer du site utilisateur vers le site de traitement.
Il en va de même du fichier résultat qui, s'il existe sera constitué sur le site de traitement et sera envoyé après exécution du programme en langage de base vers le site utilisateur.

Le schéma d'exécution qui découle de ces caractéristiques est le suivant :

1. réception d'un programme en Langage de base,
2. constitution d'un fichier de commandes (qui permettra par la suite de simuler le dialogue),
3. appel du système IDML, et en particulier du Moniteur,
4. émission des résultats.

1. Réception d'un programme en Langage de base.

Le Moniteur Général reçoit des messages successifs dont les données sont sensées constituer un programme en code exécutable.

Ces blocs de bytes vont dès lors être rangés séquentiellement dans un fichier qui pourra être exploité par le système IDML, par l'intermédiaire de l'interface Base de Données Fichiers Standard (3).

Le nom du fichier est attribué de façon standard par le Moniteur Général.

2. Constitution d'un fichier de commandes.

Pour que l'exécution du programme puisse se dérouler, le Moniteur a besoin de connaître les opérations à effectuer. Ces commandes vont être transmises par le Moniteur Général, via un fichier batch (1), au Moniteur.

Ce fichier de commandes devra contenir l'ordre d'exécution (chargement et interprétation du programme en code exécutable qui vient d'être réceptionné) et un ordre de terminaison.
Il sera désormais sous le contrôle de l'interface Terminal.

3. Appel du système IDML.

Le Moniteur Général passe le contrôle au Moniteur. La référence interne de l'utilisateur est à la disposition des différentes interfaces de Bases de Données. Les autres éléments du système ne s'intéressent pas à cette référence.

4. Emission des résultats.

Le programme en langage de base peut demander l'écriture de résultats dans un fichier (4). Le Moniteur Général doit pouvoir retrouver ce fichier afin de le transmettre à l'utilisateur. Il faut donc qu'il en connaisse le nom physique.
C'est pourquoi il est demandé à l'utilisateur de lui donner un nom de format standard.

Si ce fichier existe, le Moniteur Général va constituer une série de messages et les envoyer séquentiellement à l'utilisateur.

S'il n'existe pas, la raison peut être que l'exécution ne s'est pas déroulée convenablement.
Dans ce cas, le Moniteur aura inscrit des informations sur ce qui tient lieu d'écran : un fichier batch en sortie (2). Ce fichier sera alors envoyé à l'utilisateur.

Il reste alors à détruire toutes les traces de ce travail (c'est-à-dire tous les fichiers temporaires créés pour l'utilisateur).

3.4.2.2. Moniteur système multi-utilisateurs temps réel.

La situation du problème et les principes de réalisation ont été abordés précédemment.

Rappelons cependant que nous voulons dérouler un programme pour plusieurs utilisateurs, chacun d'eux voulant une exécution en temps réel. La configuration hardware possède une unité centrale mono-programmable.

Cette unité centrale va être allouée successivement à chaque utilisateur et ce pour une période limitée. Idéalement, cette période devrait être cadencée par une horloge. Comme ce n'est

pas possible ici, la période va être variable : il s'agit du temps d'exécution d'une action logique.

Pour que ce système fournisse des résultats positifs pour la plupart des utilisateurs, il faut que l'allocation du processeur soit équitablement réalisée. Toutes les actions devraient avoir une durée d'exécution quasiment identique.

Mais cette exigence nécessite une technique complexe pour sauver et restaurer le contexte qui peut être important.

Actuellement, cette technique n'a pas été réalisée. Une découpe assez fine des actions n'a pas pu être opérée, ce qui entraîne un déséquilibre dans le système et une dégradation globale du système.

3.4.3. Le logiciel réalisé.

3.4.3.1. Architecture du programme du site de traitement.

Elle est directement déductible de l'architecture du système IDML partagé.

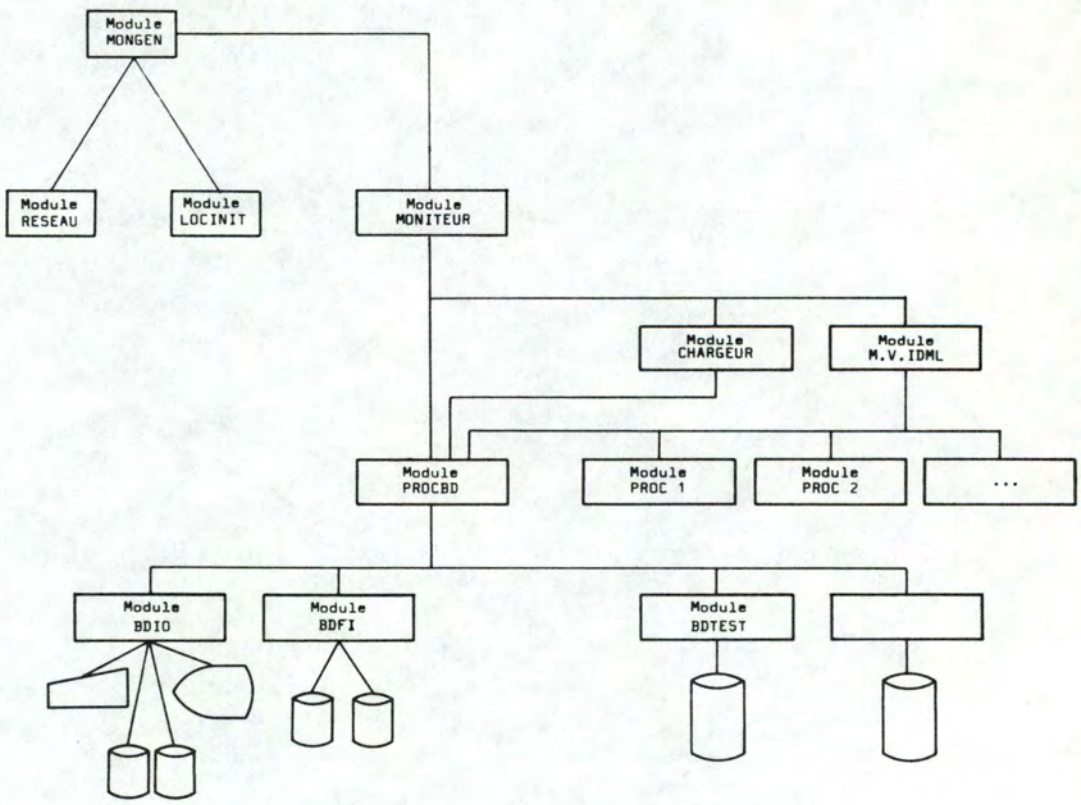


Fig. III.16. - Réalisation du niveau Lgbs.

Citons brièvement les fonctions de chacun de ces modules :

1. Le Moniteur Général.

Le Moniteur Général est chargé d'assurer le roulement des utilisateurs qui occupent l'unité centrale.
La tranche de temps allouée n'est pas fixe puisqu'elle est déterminée par l'étape à exécuter.

La découpe qui suit devrait être nettement plus fine, mais cet affinage a des conséquences sur le sauvetage du contexte à élaborer.

Déroulement du système multi-utilisateurs.

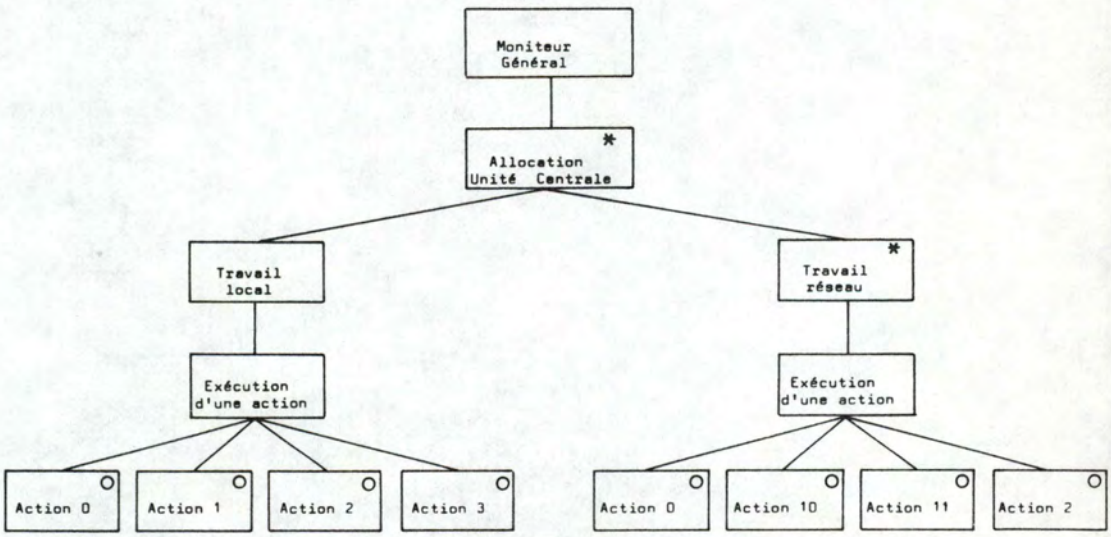


Fig. III.17. - [AP] Le programme SIDMLS.

Découpe en actions du travail de l'utilisateur local.

Voir l'arbre programmatique - Fig. III.18.

Découpe en actions du travail de l'utilisateur réseau.

Voir l'arbre programmatique - Fig. III.19.

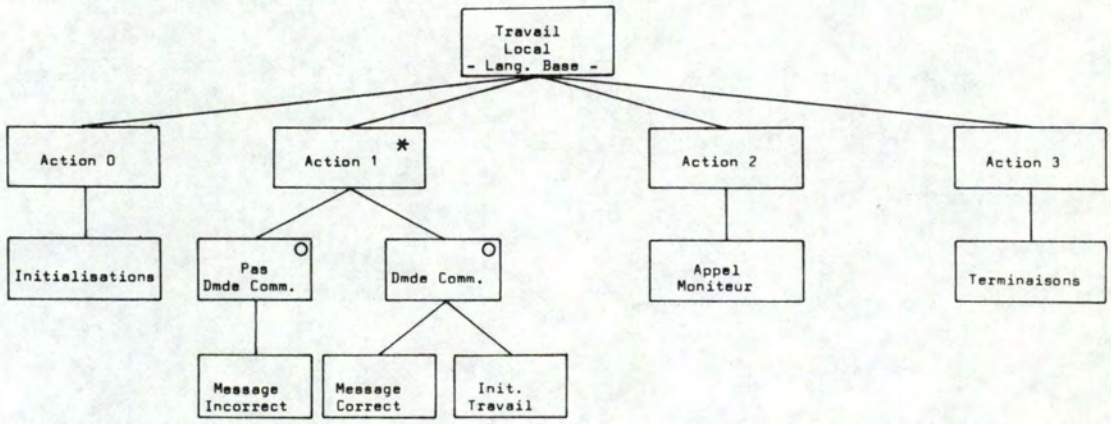


Fig. III.18. - [AP] La procédure de Travail Local.

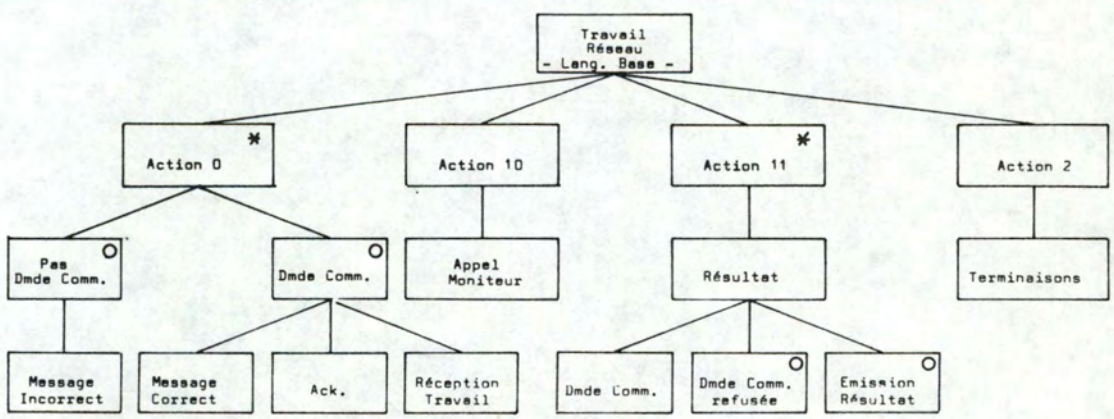


Fig. III.19. - [AP] La procédure de Travail Réseau.

2. Le module réseau.

C'est à lui qu'incombe le rôle de préparer le travail pour le système IDML. Le fait que l'utilisateur n'est pas l'utilisateur local doit être transparent au Système IDML.

En ce qui concerne la liaison avec le réseau, les procédures sont identiques. Elles concernent la prise en charge (lecture, écriture) d'un message du réseau.

Au niveau de l'application, le module remplit trois fonctions :

- il constitue le fichier qui contient le programme en code exécutable envoyé par l'utilisateur. Ce fichier reçoit un nom de forme standard FEXEC‹adresse de l'utilisateur›.E

- il crée un fichier de commande. Du point de vue du système IDML, ce fichier constitue le clavier du terminal. Les commandes vont dès lors être rangées dans le fichier batch en entrée;

- il renvoie à l'utilisateur le fichier de résultat, si celui-ci existe. Il porte un nom convenu : DTCOM‹adresse réseau utilisateur›.
  Si ce fichier n'existe pas, il renverra le fichier qui tient lieu d'écran : le fichier batch en sortie.

  Enfin, il reste à détruire tous les fichiers qui ont servis à faire le travail :
  - le fichier contenant le code exécutable,
  - le fichier de résultats,
  - le fichier batch en entrée,
  - le fichier batch en sortie,

3. Le module Locinit.

Ce module permet à l'utilisateur local de signaler qu'il veut travailler. Ensuite, il sera pris totalement en charge par le système IDML.

4. Le processeur Base de Données.

Il supervise les appels aux différentes interfaces :
- l'interface BDIO,
- l'interface BDFI,
- l'interface BDTEST.

5. Le Chargeur et la Machine Virtuelle.

Ils mettent en pratique les concepts que nous avons exposés dans la partie 2 (- Le système IDML mono-utilisateur -).

3.4.3.2. Un langage d'assemblage.

Quand l'utilisateur travaille directement en code intermédiaire, il doit fournir au système IDML un programme en code exécutable stocké dans un fichier.

Un outil doit lui être donné afin de pouvoir constituer physiquement ce fichier. Plutôt que de lui faire écrire une suite d'octets, un langage d'assemblage rudimentaire a été établi.

L'utilisateur peut dès lors créer, à l'éditeur, un fichier lisible, contenant son programme. Quatre mots réservés lui permettent de diriger l'assemblage :

- INSTRUCTIONS
- DESCRIPTEURS
- VALEURS
- ZONE LIBRE

Les codes des instructions sont donnés sous forme mnémonique plutôt que numérique; les arguments sont des adresses de descripteurs ou des valeurs immédiates.

Les descripteurs sont constitués d'une suite de mots standard de 32 octets. Chaque mot peut contenir plusieurs informations. Le calcul sera effectué par le programme d'assemblage sur base du type de mot signalé : M1, M2, M3, M5.

Les valeurs sont entrées par octet, qui est le mot de base pour la zone de valeur.

Enfin, la zone libre est constituée du nombre donné d'octets.

Des commentaires peuvent être introduits par l'utilisateur après le symbole spécial #.

P A R T I E 4. LE SYSTEME IDML DISTRIBUE.

4.1. OBJECTIFS ET PRINCIPES.

Introduction.

Si le système IDML mono-utilisateur et le système IDML partagé ont été non seulement spécifiés mais aussi implémentés, il n'en est pas de même pour le système IDML distribué.

Son étude est à l'état de recherche.

Pour commencer, nous rappellerons l'objectif poursuivi et nous le situerons dans la démarche de travail (paragraphe 4.1.1.). Ensuite, nous re-citerons quelques problèmes soulevés dans le chapitre 1.2. (paragraphe 4.1.2.) et nous situerons notre position vis-à-vis de ces problèmes (paragraphe 4.1.3.).

4.1.1. Objectif du système IDML Distribué.

Nous avons vu, dans la partie précédente qu'un site sur le réseau pouvait remplir deux fonctions :

1. stocker des données et fournir l'accès à ces données;
2. permettre l'accès à des données sur un autre site.

Le système ne réalisant que la première fonction est appelé "système partagé". Les utilisateurs sont de deux types :

- utilisateur local : il travaille comme si le système était mono-utilisateur;
- utilisateur réseau : l'interaction entre l'utilisateur et le système partagé se fait par l'intermédiaire d'un programme d'application particulier et du réseau.

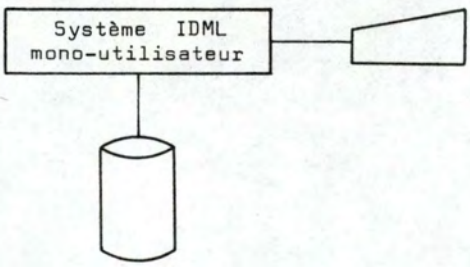
Maintenant, nous voudrions que le système remplisse la deuxième fonction.

> permettre à l'utilisateur local d'accéder, via le système IDML, à la totalité des données disponibles sur les sites de stockage du réseau.

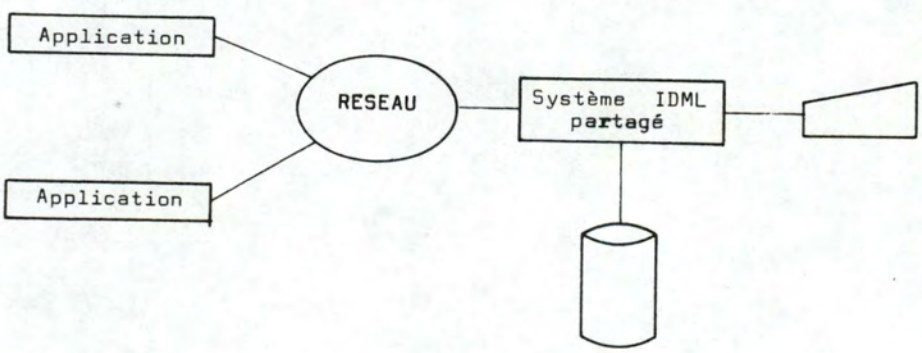
Nous dirons alors que le système IDML est distribué.

Reprenons schématiquement l'évolution du système IDML :

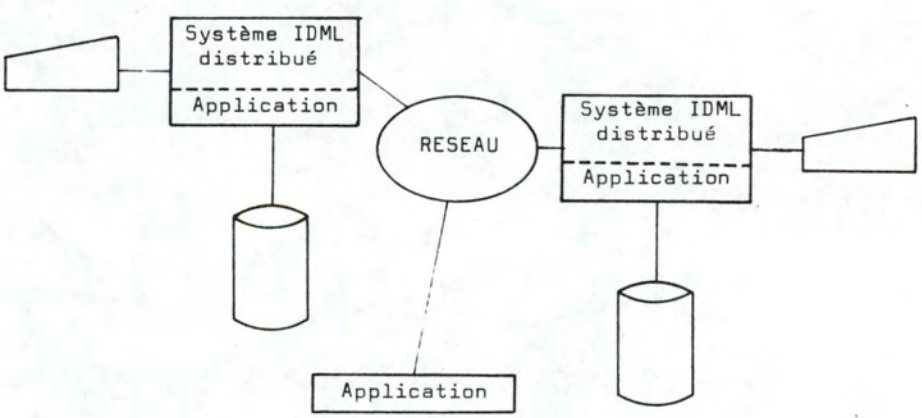
1. Système IDML mono-utilisateur. - Fig. IV.1.



2. Système IDML partagé. - Fig. IV.2.



3. Système IDML distribué. - Fig. IV.3.



4.1.2. Quelques problèmes de la distribution.

Dans le chapitre portant sur l'implémentation d'un système de Gestion de Bases de Données Réparties (chapitre 1.2. - Implémentation d'un SGBDR -), nous nous sommes intéressés à cinq aspects particuliers.

1. L'architecture générale du système doit-elle tenir compte ou non des systèmes locaux préexistants ?

2. Quels sont les problèmes liés au contrôle des données (éclatement de fichiers, copies multiples) ?

3. Comment retrouver et manipuler des données; où se trouve le catalogue ?
   - sur un site particulier,
   - sur chaque site, en entier,
   - sur chaque site, partiellement.

4. Quel est le langage de la communication ?
   - langage de haut niveau de Manipulation des données,
   - langage intermédiaire,
   - primitive Base de Données.

5. Quel est le protocole de communication sur le réseau ?

D'autres problèmes n'ont pas été abordés. Ils concernent, entre autres,

le contenu de la communication : (quelles sont les règles qui permettent d'éclater convenablement un travail en diverses requêtes adressées à des sites extérieurs; quels sont les éléments à leur transmettre) et

la gestion du travail (synchronisation des requêtes et des réponses, regroupement d'information venant de divers sites,...)

4.1.3. Les limites de cette recherche.

Pour pouvoir résoudre totalement les problèmes de la distribution, il faudrait se plonger bien plus dans ce domaine.

Nous allons dès lors nous restreindre à certains éléments et voir comment nous pouvons les intégrer au système IDML partagé (réponse au problème 1).

Nous laisserons de côté le contrôle des données (réponse au problème 2).

Nous posons l'hypothèse que le système connaît la localisation des données (réponse au problème 3).

Quant à la communication en elle-même, nous chercherons comment travailler aux trois niveaux de dialogue déjà établis dans le système partagé (réponse au problème 4).

Le protocole de communication ne subira, lui, aucune modification fondamentale par rapport à celui établi pour le système partagé (réponse au problème 5).

Nous aborderons enfin de façon succincte l'organisation du travail sous un angle particulier : comment le système IDML distribué peut-il gérer efficacement la présence d'un utilisateur local qui travaille sur le site local mais peut avoir besoin de travailler avec d'autres sites sur le réseau et des utilisateurs externes.

4.2. RECHERCHE DE SOLUTION.

Introduction.

Nous basant sur l'architecture du système IDML partagé, nous avons cherché à l'étendre à un système distribué. Pour ce faire, de nouveaux modules semblent nécessaires.

Plusieurs solutions sont à envisager :

- ajouter un processeur spécialisé à la Machine Virtuelle (paragraphe 4.2.1.),
- ajouter une interface Base de Données particulière (paragraphe 4.2.2.),
- étendre les interfaces Base de Données (paragraphe 4.2.3.).

Un aperçu de l'organisation interne suite à ces modifications fera l'objet d'un dernier point (paragraphe 4.2.4.).

4.2.1. Extension de la Machine Virtuelle.

L'extension de la Machine Virtuelle se fait par l'adjonction d'un processeur spécialisé.
Ce processeur sera déclenché par de nouvelles instructions du langage. Il aura pour objet la communication avec le réseau.

Proposition d'architecture. voir Fig. IV.4.

Ce module permet le dialogue aux trois niveaux retenus.

1. primitive :

   l'instruction est similaire à celle qui déclenche le processeur Base de Données. Outre l'adresse des arguments, elle contient l'adresse physique du site réseau qui pourra traiter la demande.

   Le processeur de communication se chargera de préparer le(s) message(s) qui constitue(nt) la primitive, de l'envoyer, de réceptionner la réponse et de la transmettre à la Machine Virtuelle.

2. langage intermédiaire :

   l'ensemble des instructions des descripteurs et des valeurs qui constituent le programme à transmettre se trouve dans un fichier. L'instruction peut dès lors consister en un ordre au processeur de communication d'envoyer ce fichier et de réceptionner le résultat dans un autre fichier.

L'instruction contiendrait au moins trois informations : le nom du fichier à envoyer, le nom du fichier de réception et l'adresse physique du correspondant.

3. langage IDML :

le principe est le même que pour le langage intermédiaire. Seul le contenu du fichier à envoyer est différent.

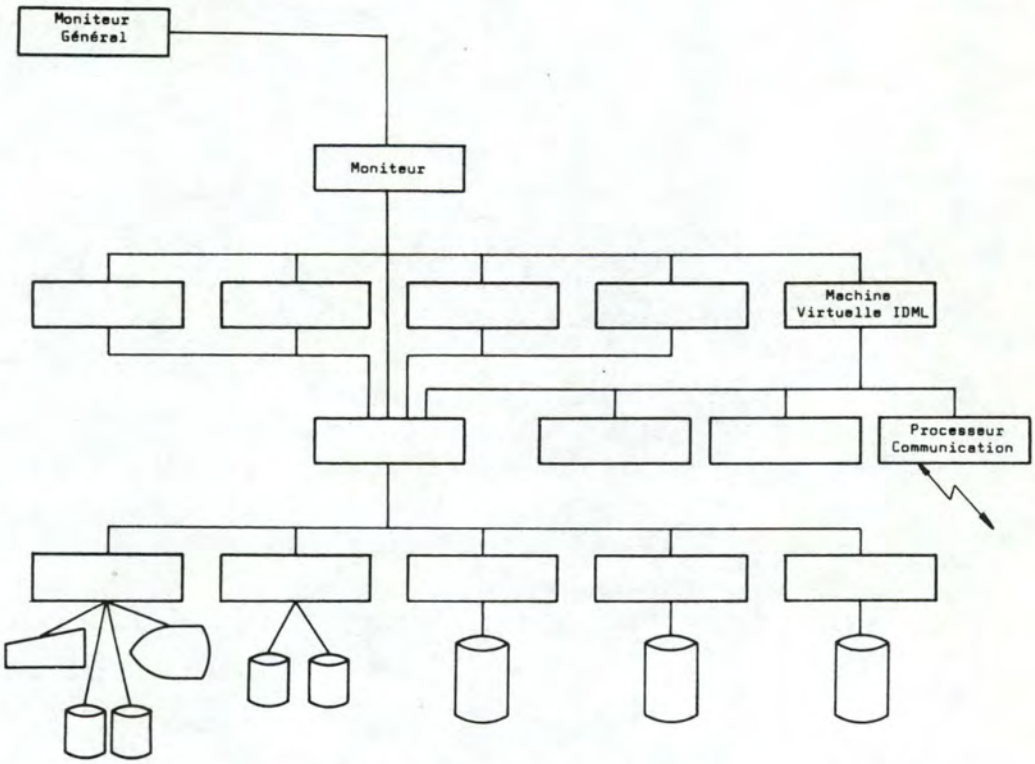


Fig. IV.4. - Le système IDML distribué : proposition 1.

Dans un système IDML complet, ces instructions pourront être générées par le compilateur qui constituera également le fichier à envoyer et le placera sous le contrôle de l'interface BDFI.
De même qu'il a besoin de l'interface Base de Données Schémas concernant les Bases de Données locales, le compilateur devra accéder au catalogue réseau afin de connaître l'adresse (ou les adresses) du (des) site(s) de traitement.

4.2.2. Une interface Base de Données particulière.

Cet interface regroupe, logiquement, toutes les Bases de Données qui se trouvent sur d'autres sites. Elle aura pour but de transmettre une primitive qui ne correspond pas à une Base de Données locale à un site du réseau.

Proposition d'architecture. voir Fig. IV.5.

Ce module permet exclusivement le dialogue au niveau primitive.

L'appel à cette interface se fait comme l'appel à toute autre interface. Les Bases de Données qui sont sous son contrôle sont identifiées par un code SREF différent de toutes les références SREF locales.

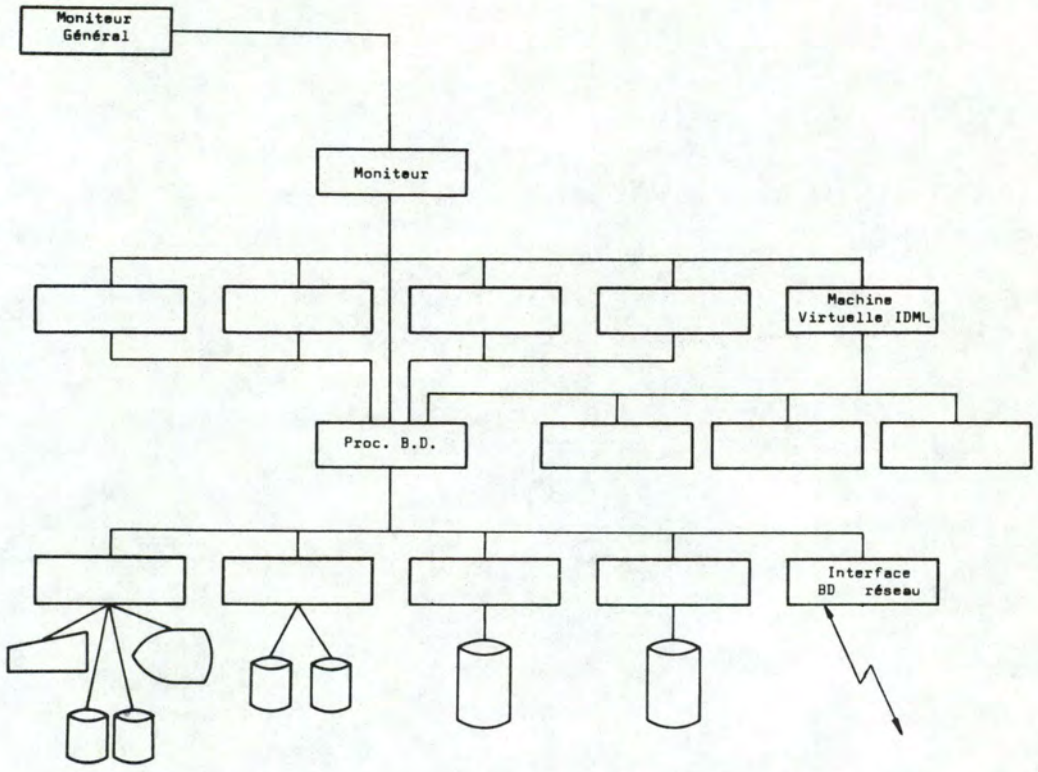


Fig. IV.5. - Le système IDML distribué : proposition 2.

Il faut donc standardiser l'attribution du code SREF par le processeur Base de Données.

SREF est composé de deux éléments :

1. ID_SITE : adresse physique du site où réside la Base de Données,

2. ID_BD : référence interne de la Base de Données sur ce site

Si l'utilisateur connaît le site sur lequel est stockée la Base de Données à laquelle il veut accéder, il peut le signaler, afin de faciliter la tâche du processeur Base de Données, lors de la première primitive qu'il adresse :

SREF aura la valeur ID_SITE + 0

Si l'utilisateur ne le connaît pas, le processeur Base de Données interrogera successivement les différentes Bases de Données locales, puis les différents sites du réseau, jusqu'au moment où une réponse positive sera obtenue ou que les possibilités seront épuisées.

4.2.3. Extension d'une interface Base de Données.

L'interface a actuellement pour rôle de gérer une Base de Données. Dans un contexte distribué, cette Base de Données peut être en relation avec d'autres Base de Données (éclatement d'un fichier, copies multiples).

Des opérations pourraient s'effectuer entre ces Bases de Données, de manière transparente pour l'utilisateur.
Elles seraient dès lors prises en charge par l'interface de la Base de Données.

Proposition d'architecture.

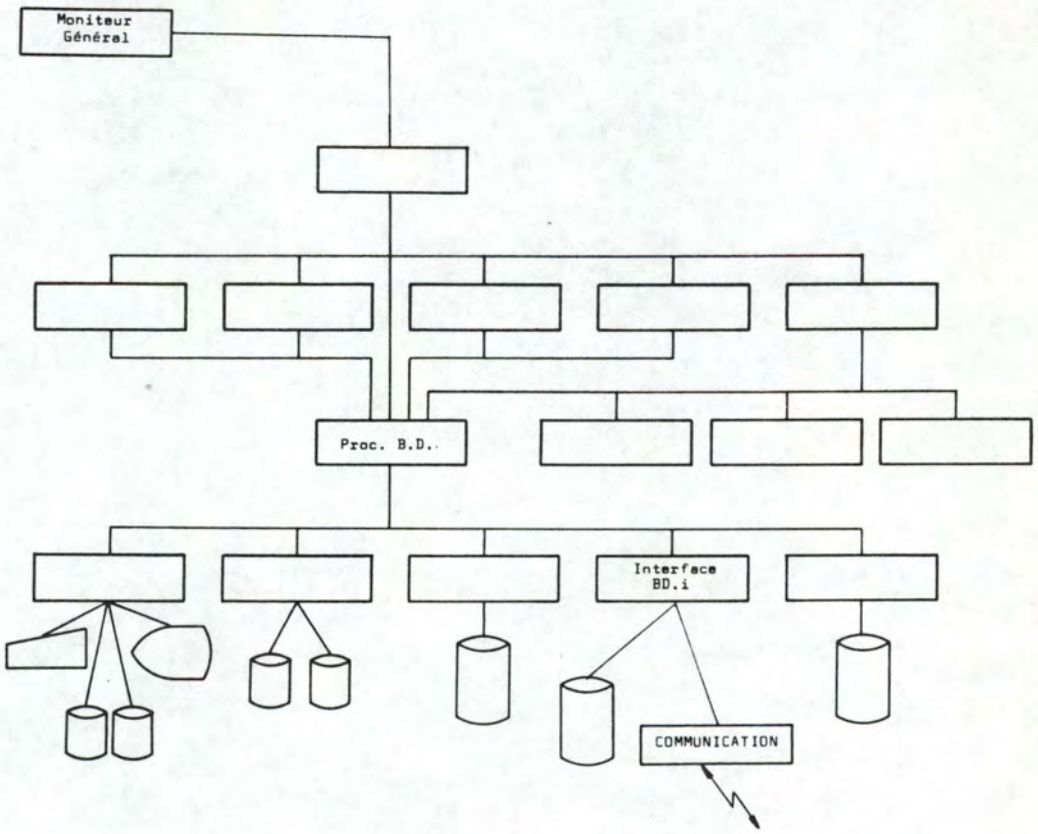


Fig. IV.6. - Le système IDML distribué : proposition 3.

Cette structure force l'utilisateur à ignorer tout du contexte distribué, car il n'a aucune prise dessus.
C'est uniquement l'interface qui gère l'appel à la Base de Données locale et/ou à des Bases de Données sur le réseau.

A noter que cette architecture pourrait apporter une aide en ce qui concerne le contrôle des données : les Bases de Données doivent pouvoir communiquer entre elles afin de régler les problèmes de cohérence.

4.2.4. Organisation interne.

Il faut que le système puisse supporter plusieurs utilisateurs simultanés et tendre vers une exécution en temps réel.

Trois principes ont dirigé l'élaboration du système partagé dans cette optique :

1. aucun utilisateur n'est prioritaire par rapport aux autres. Le système s'occupe successivement de chacun d'eux.

2. le processus (exploitation du système IDML) est subdivisé en étapes logiques. Chaque fois qu'un utilisateur est pris en charge, une seule étape est exécutée.

3. la découpe en étapes suit deux règles :
   - ne jamais être inactif à cause du réseau,
   - toutes les étapes ont besoin d'un temps similaire.

Nous avons vu que cette dernière règle n'a pas pu être réalisée totalement (point 3.4.3.2. - Implémentation du niveau Lgbs) pour des raisons de sauvetage de contexte.

Ce problème est encore plus aigu ici.

En effet, dans le système partagé, la gestion des utilisateurs incombait totalement au Moniteur Général. Ce n'est plus le cas dans le système distribué vu que des appels au réseau peuvent se faire à tout moment à partir du système lui-même, sans que le Moniteur Général en ait connaissance.

Le sauvetage du contexte ne peut donc pas être mis uniquement sous son contrôle.

Chaque processeur de communication doit rompre une étape, signaler au Moniteur Général cette rupture et lui indiquer le contexte à sauver (contexte qui sera différent suivant la situation du processeur dans l'architecture).

Les étapes ne seront donc pas statiques, mais doivent être gérées dynamiquement, suivant qu'il y a une communication ou non avec le réseau.

P A R T I E 5. EVALUATION DU SYSTEME IDML PARTAGE.

Introduction.

L'exploitation du système IDML partagé nécessite la participation de trois éléments :

- des utilisateurs sur le réseau, avec un programme d'application particulier
- un site supportant le système IDML
- le réseau.

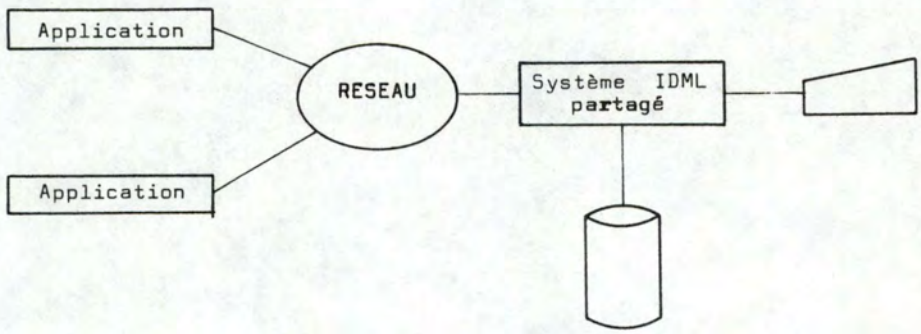


Fig. V.1. - Le système IDML partagé.

Pour chacun de ces éléments, nous allons comparer l'emploi des différents niveaux de travail.

- primitives
- langage de base
- langage IDML.

Enfin, nous essayerons de tirer quelques conclusions générales d'utilisation du système IDML partagé, sur base de cette évaluation.

5.1. EVALUATION POUR L'UTILISATEUR.

Cette évaluation va se faire en trois temps :

- tout d'abord, quelles facilités d'utilisation apporte chaque niveau de langage pour l'utilisateur,
- ensuite, quels sont les éléments ou modules qui constituent l'architecture du programme d'application,
- enfin, à quel temps de réponse l'utilisateur doit-il s'attendre.

5.1.1. Facilités et efficacité d'utilisation.

Dans l'étude et la réalisation que nous avons faites, nous avons posé une hypothèse qu'il est nécessaire de rappeler ici : l'utilisateur qui travaille au niveau primitive le fait de façon totalement interactive.

Si l'utilisateur s'exprime par primitives , cela nécessite une interaction constante entre l'homme et la machine. Il doit en outre avoir une connaissance précise du modèle des accès logiques, tant en ce qui concerne les objets qu'il veut manipuler, que les primitives qui lui permettent cette manipulation.

S'il est vrai que certaines simplifications ont été introduites dans ce modèle (par exemple l'ouverture automatique des fichiers), pour rendre certaines notions transparentes à l'utilisateur, il n'en reste pas moins que l'utilisateur doit être familiarisé avec cette méthode de travail.

Ce travail totalement interactif permet évidemment de réagir très vite à une erreur, mais dans la réalité il semble difficile de l'appliquer.

Par contre, le fait d'intégrer ces primitives dans un langage conventionnel paraît être réaliste. En effet, l'avantage lié au traitement d'erreur est conservé, pour autant que le programme soit bien écrit.
Quant à la lourdeur de traiter interactivement avec des primitives (paramètres, codes, lenteur...), elle s'intègre psychologiquement mieux dans un programme qui, en plus, pourra prendre en charge certaines manipulations fastidieuses.

Le traitement batch a, quant à lui, l'inconvénient d'être fermé aux erreurs puisque le contenu ne peut être qu'une suite de primitives où tout a été décidé d'avance. Ce qui s'oppose à un traitement dynamique.

A l'opposé, si l'utilisateur travaille en langage IDML , il spécifiera uniquement les critères de recherche, et ne s'occupera nullement de la gestion proprement dite de la Base de Données. Le langage, bien que performant, est néanmoins simple d'usage.
L'utilisateur ne doit donc pas être un spécialiste pour travailler sur des Bases de Données.

Entre ces deux langages, l'utilisateur peut encore s'exprimer directement dans le langage interne.

Cette méthode ne s'adresse pas à l'utilisateur occasionnel, car elle nécessite des connaissances précises en programmation, et surtout une connaissance parfaite du système : l'utilisateur doit fournir un programme en code exécutable.
Par contre, elle permet de construire des "super-primitives" ou des macros, technique qui peut s'avérer souvent très utile.

En résumé :

Il ne fait pas de doute que le langage le plus aisé à utiliser est le langage IDML. Quant aux deux autres (primitives et langage interne), l'utilisateur inexpérimenté se trouvera plus à l'aise avec l'emploi des primitives, tandis que le programmeur trouvera un plus grand bénéfice à construire un programme en langage interne, travail quelque peu ardu mais qui lui permettra un traitement plus efficace. En effet, une fois le programme écrit, c'est le système qui prend en charge tout le traitement. L'utilisateur n'a qu'à recevoir les résultats.

- pour l'utilisateur occasionnel : IDML > Primitives > Lgbs
- pour l'utilisateur expérimenté : IDML > Lgbs > primitives.

5.1.2. Taille du programme d'application.

1. Au niveau primitives.

   Le programme d'application se compose de deux modules :

   - un module réseau :
     qui se charge du protocole de communication et de l'émission/réception de valeurs (ici, les arguments d'une primitive),

   - un module travail interactif :
     qui permet à l'utilisateur de fournir les valeurs des arguments et de recevoir les résultats.

2. Au niveau langage de base.

   Nous trouvons ici aussi deux modules :

   - un module réseau :
     chargé du protocole et de l'émission/réception de blocs de caractères

   - un module de transfert de fichiers :

     - le programme exécutable à transférer se trouve dans un fichier. Il faut le scinder en blocs de caractères;
     - les résultats qui reviennent par le réseau doivent être rangés dans un fichier.

   En outre, le système doit comprendre un logiciel d'assemblage :
   il permet à l'utilisateur de constituer le fichier comprenant le programme en code exécutable.

3. Au niveau langage IDML.

Toujours deux modules :

- le module réseau,

- le module de transfert de fichiers

  - le fichier source vers le site de traitement,
  - le fichier résultat.

Le fichier source est un fichier de texte, mais qui comporte, en plus du programme lui-même, une en-tête (descripteur du fichier) conforme au système IDML. Il faut donc un logiciel éditeur pour construire cette en-tête particulière et accepter le texte source.

En résumé :

La taille du programme d'application ne joue pas de rôle pour classifier les niveaux de langage. Quel que soit ce langage, la taille du programme d'application est similaire.

Remarque :

Si le site qui comporte le programme d'application supporte également un système IDML mono-utilisateur, celui-ci peut évidemment être utilisé pour construire le programme en code exécutable, mais à une condition: il faut que le compilateur ait à sa disposition la Base de Données des Schémas correspondant au programme, et que le compilateur soit correct pour le site de traitement.

5.1.3. Le temps de réponse.

Ce temps de réponse est fonction de plusieurs éléments :

- du travail demandé,

- du temps de transfert sur le réseau qui est fonction de la taille et du nombre des messages et du trafic sur le réseau,

- de la charge du site de traitement qui induit le temps d'attente.

Le temps d'attente dépend fortement de l'implémentation, comme l'a montré l'exemple des 10 utilisateurs (chapitre 3.3.)

Dans le cas présent, cette implémentation considère le chargement et l'exécution d'un programme en langage interne comme une étape à réaliser sans interruption.
D'autre part, les utilisateurs sont traités dans un ordre pré-établi.

Faisons une hypothèse sur la charge du site de traitement :

soit n utilisateurs simultanément au travail.

1. Au niveau primitives.

A ce niveau, le protocole nécessite deux communications (demande de communication et fin de communication) qui correspondent toutes deux à une étape sur le site de traitement. Supposons en outre que le travail comporte p primitives.

Suivant la découpe en étapes, nous pouvons dire que le système IDML exécutera (p+2) étapes pour cet utilisateur.

Comme entre chacune des ces étapes, le système traitera n-1 utilisateurs, le temps maximum d'attente sera de (p+1) * T{n-1}

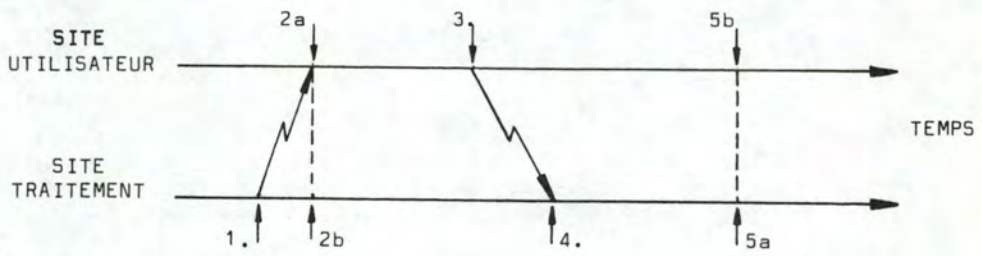
T{n-1} représente le temps de traitement d'une action pour chacun des n-1 utilisateurs. Ce temps est fonction de l'état de travail dans lequel se trouve chacun des utilisateurs.

Nous avons dit 'temps maximum'. En effet, dans la réalité, ce temps peut être inférieur car un certain parallélisme existe du fait que nous travaillons sur deux sites distincts.

Regardons ce qui se passe à un moment donné sur chacun des deux sites:

- l'utilisateur vient de recevoir la réponse à une primitive. Le programme d'application lui communique le résultat via l'écran du terminal. L'utilisateur va alors proposer la primitive suivante : pour cela, il va donner de nouvelles valeurs pour certains paramètres que le programme d'application transforme en un message qu'il émet sur le réseau.

- le site de traitement a envoyé une réponse sur le réseau. Il peut alors commencer à traiter les utilisateurs suivants.



1. émission de la réponse
2a la réponse est disponible
2b début du traitement de l'utilisateur suivant
3. émission de la requête-primitive
4. la requête est en attente
5a début de la période de traitement de l'utilisateur

de 2b à 5a traitement des (n-1) utilisateurs

de 3. à 5b temps d'attente

Fig. V.2. - Parallélisme site utilisateur / site traitement.

Dès lors, pour l'utilisateur, une fois sa requête envoyée, le temps d'attente se situe dans l'intervalle [0 .. T{n-1}].

Le temps de réponse =

exécution de p primitives

+ transfert (aller et retour) de (p+2) messages

+ attente variant de [0 .. T{n-1}*(p+1)]

2. Au niveau langage de base.

Le trafic sur le réseau comporte quatre messages de protocoles
(demande et fin de communication de l'utilisateur vers traitement;
(demande et fin de communication du site traitement vers l'utilisateur)
et des messages de blocs de caractères.

Quelle que soit la longueur du programme en code, le système exécutera trois étapes :

- réception du programme,
- exécution du programme
- émission des résultats.

Nous posons l'hypothèse que le site utilisateur est prêt à recevoir les résultats, sans quoi le site de traitement devrait essayer plusieurs fois, et entre ces essais traiterait les autres utilisateurs.

Le Temps de réponse =

exécution de p primitives

+ transfert des messages de protocole, du fichier exécutable et du fichier de résultats

+ attente variant de [0 .. T{n-1}] pour le début de la première étape, plus 2*T{n-1} pour les deux autres étapes.

5.2. EVALUATION DU SITE DE TRAITEMENT IDML PARTAGE.

5.2.1. Taille du système.

La taille du logiciel qui réalise le système IDML partagé est lié au(x) langage(s) autorisé(s) : IDML > Lgbs > Primitives.

1. Au niveau primitives

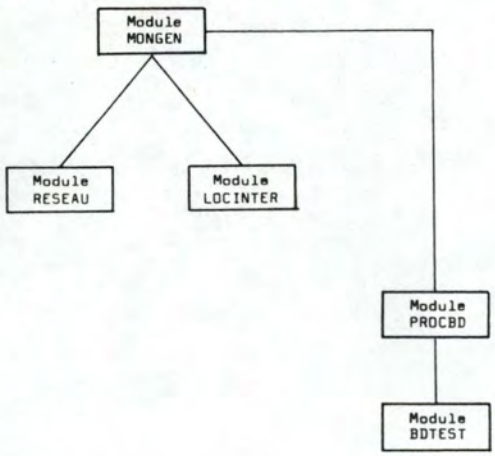


Fig. V.3. - Réalisation du niveau Primitives.

Voici un ordre de grandeur de la taille des modules :
[voir Annexe IV]

- module MONGEN : code = 6 Kb; données = 4 Kb
- module RESEAU : code = 10 Kb; données = 2 Kb
- module LOCINTER : code = 17 Kb; données = 1 Kb
- module PROCBD : code = 1 Kb;
- module BDTEST : code = 12 Kb; données = 1 Kb

A ces chiffres, il faut encore ajouter le code des procédures du système qui sont en librairie ainsi qu'une zone de mémoire utilisée principalement par le SGF.

2. Au niveau Langage interne

La taille des différents modules est la suivante :

- module MONGEN : code = 6 Kb; données = 4 Kb
- module RESEAU : code = 12 Kb; données = 2 Kb
- module LOCINTER : code = 17 Kb; données = 1 Kb
- module MONITEUR : code = 3 Kb;
- module CHARGEUR : code = 2 Kb;
- module M.V.IDML : code = 4 Kb; données = 4 Kb
- module PROC.1 : code = 6 Kb;
- module PROC.2 : code = 6 Kb;

- module PROC.3 : code = 5 Kb;
- module PROCBD : code = 1 Kb;
- module BDIO : code = 6 Kb;
- module BDFI : code = 5 Kb;
- module BDTEST : code = 12 Kb; données = 1 Kb

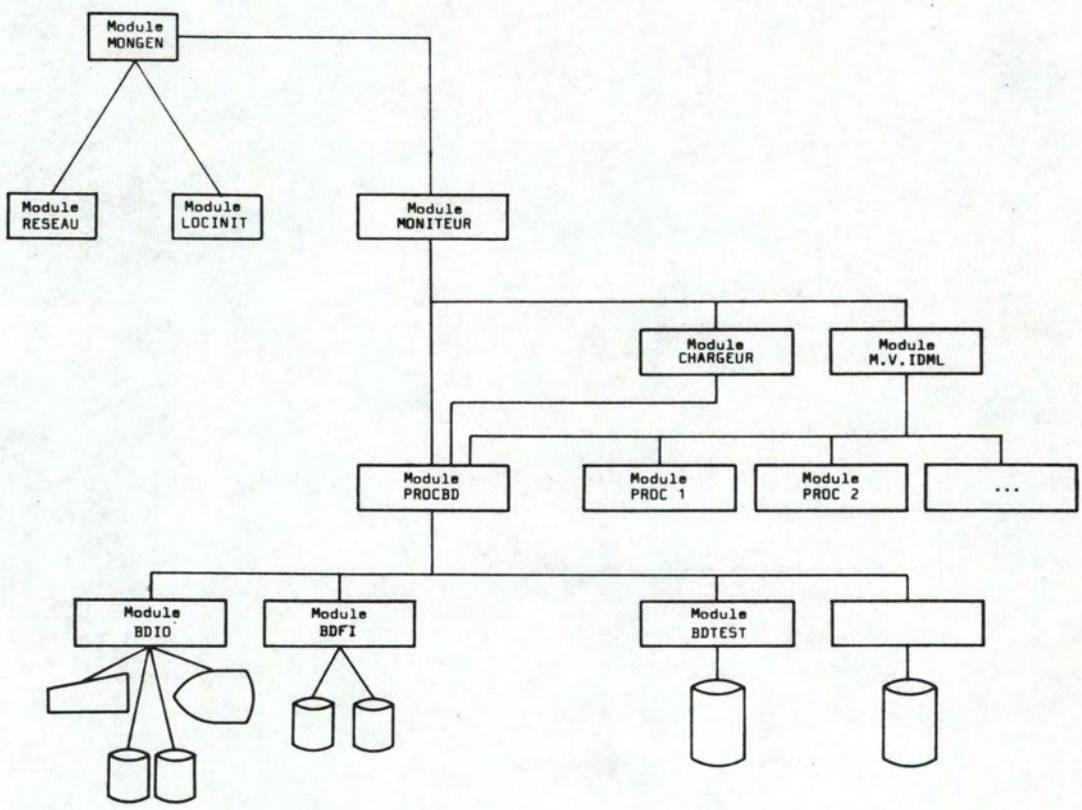


Fig. V.4. - Réalisation du niveau Lgbs.

3. Au niveau langage IDML

voir Fig. V.5. (- page suivante -)

5.2.2. Charge du système.

La charge du système est fonction :

1. du nombre d'utilisateurs simultanément au travail,
2. du travail demandé par chaque utilisateur.

Evaluons le temps de traitement global utilisé par un utilisateur du réseau.

1. Au niveau primitives.

Le traitement se décompose comme suit :

1. traitement proprement dit des primitives (travail du processeur Base de Données et des interfaces Bases de Données conventionnelles);

2. traitement lié au protocole de communication :
   - réception sans attente d'un message,
   - informations à reconnaître en début de chaque message;
   - informations à préparer pour des messages à envoyer.

3. traitement d'initialisation et de clôture d'un travail.

(de 1 à 3, l'importance en temps nécessaire décroît).

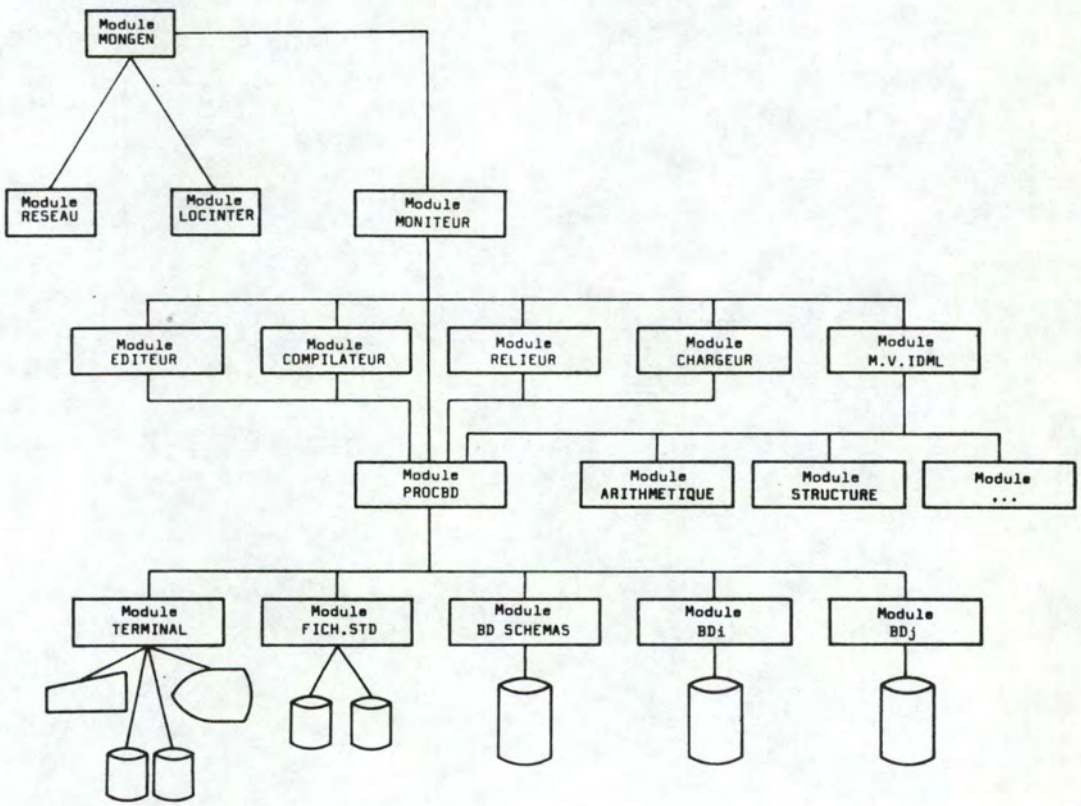


Fig. V.5. - Projet de réalisation du niveau IDML.

2. Au niveau langage interne.

Quatre types de traitement ont été retenus :

1. traitement proprement dit : chargement et exécution du programme en code interne;

2. traitement lié aux transferts de fichiers :
   - le programme en code doit être stocké dans un fichier et un fichier de commandes doit être créé;
   - les résultats à émettre se trouvent dans un fichier;

   Il nécessite donc de nombreux accès disques, ce qui est coûteux en temps.

3. traitement lié au protocole de communication :
   - réception avec attente (généralement nulle) d'un message;
   - reconnaissance d'informations et
   - informations à préparer (en-têtes des messages).

4. traitement d'initialisation et de cloture.

3. Au niveau langage IDML.

1. traitement proprement dit : compilation, reliage, chargement et exécution du programme;

2. 3. 4. idem langage interne.

Il apparaît donc nettement que des primitives au langage IDML, le temps nécessaire pour traiter un travail est croissant.

5.2.3. Un utilisateur particulier : l'utilisateur local.

Deux remarques doivent être faites concernant cet utilisateur :

- la première concerne son temps d'attente;
- la seconde concerne l'influence qu'il a sur le système.

Rappelons qu'actuellement cet utilisateur est unique.

1. Temps d'attente (niveau primitives).

Au paragraphe 5.1.3., nous avons vu qu'un certain parallélisme était possible. Ce n'est plus réalisable ici, vu que le travail interactif avec l'utilisateur se fait sur le même site que le traitement des autres utilisateurs.

Le temps d'attente sera donc toujours de (p-1)*T(n-1)

2. Influence de l'utilisateur local.

1. Au niveau primitives :

L'interaction entre l'utilisateur et le système se fait via l'écran et le clavier.

- l'écran pour visualiser le résultat de l'exécution d'une primitive;
- le clavier et l'écran pour entrer des valeurs pour les arguments d'une primitive.

Ces opérations entraînent une diminution des performances globales pour trois raisons :

- l'utilisateur est lent : le temps de réflexion pour entrer des valeurs est élevé;

- les opérations de lecture et écriture sur un terminal sont des opérations d'Entrée/Sortie qui prennent du temps; (actuellement, la gestion d'écran se fait par programme Pascal).

- ces opérations d'Entrée/Sortie sont difficilement interruptibles par le réseau.

2. Au niveau langage interne

A ce niveau par contre, le travail de l'utilisateur local exige moins de temps, car il n'y a pas lieu de faire des transferts de fichiers.

5.3. EVALUATION DU TRAFIC SUR LE RESEAU.

Ce trafic a été évalué pour différentes transactions :

1. la création d'un article;
2. la lecture du Ième article dans un fichier séquentiel;
3. la modification ou la suppression du Ième article
4. la sélection de k articles dans un fichier de n articles.

Pour chacune de ces transactions :

1. nous verrons la suite des primitives à utiliser;

2. nous écrirons un programme en code interne minimum (nous n'exploiterons pas les possibilités du langage en ce qui concerne les opérations d'impression de résultats, et nous limiterons les tests d'erreurs) qui réalise la même transaction.

L'évaluation va être fait en nombre de caractères émis sur le réseau, et en nombre d'accès au réseau.
Pour plus de détails sur les chiffres qui vont suivre, se référer à l'annexe programme [annexe III.].

Messages de protocole.

- demande de communication : 2 octets / Dmde-Comm
- fin de communication, ack : 1 octet / Fin-Comm, Ack
- en-tête d'un message primitive :
  - le premier message : 8 octets / En-tête PM
  - le second message : 2 octets / En-tête SM
- en-tête d'un message bloc de caractères : 2 octets / En-tête BC

Remarque : le premier message primitive peut recevoir 242 octets de l'utilisateur, le second 248.
Un message bloc de caractères peut en contenir 248.

Arguments d'une primitive.

- Z_CODES : 30 octets
- Z-IDENT : 90 octets
- Z-VALUE, RFIELD : 256 octets

Langage interne.

- instructions : 1 à 3 mots de 32 bits, soit 4 à 12 octets
- descripteurs : 2 à 3 mots de 32 bits, soit 8 à 12 octets

Fichier contenant un programme en code.

- descripteur général : 5 octets
- descripteur de segment : 7 octets

5.3.1. La création d'un article.

1. Au niveau primitives.

   les opérations :

   - ouvrir la Base de Données (Z_CODES, Z_VALUE)
   - écriture d'un article (Z_CODES, Z_VALUE)
   - fermeture de la Base de Données (Z_CODES).

   le protocole :

   - une demande de communication / Acknoledgement
   - une fin de communication / Acknoledgement.

   le trafic :

   protocole : Dmde-Comm + Ack + Fin-Comm + Ack ==> 5 octets

   primitive 1: En-tête PM + Z-CODES + Z-IDENT
   * 2 (requête/réponse) ==> 256 octets

   primitive 2: En-tête PM + En-tête SM
   + Z-CODES + Z-VALIT
   * 2 ==> 592 octets

   primitive 2: En-tête PM + Z-CODES
   * 2 ==> 74 octets

   soit un total de 12 accès au réseau pour le transfert de 927 octets.

2. Au niveau langage interne.

   les instructions :

   - ouvrir la Base de Données
     - maj de COP, SREF, PROTECT
     - appel du processeur BD

   - écrire un article
     - maj de COP, COREC
     - appel du processeur BD

   - fermer la Base de Données
     - maj de COP
     - appel du processeur BD

   Le nom de la Base de Données, le mot de passe et les valeurs d'items de l'article sont supposés être stockés dans la mémoire lors du chargement (valeurs initialisées).

les descripteurs :

Z-CODES, Z-IDENT, Z-VALIT
COP, SREF, PROTECT, COREC

les valeurs :

une zone pour Z-CODES, pour Z-IDENT et pour Z-VALIT

le trafic :

protocole : Dmde-Comm + Ack + Fin-Comm + Ack
* 2 (2 communications) ==> 10 octets

transfert fichier code :

- descripteur général et
  4 descripteurs segments ==> 33 octets
- instructions (6 MOVE + 3 CALL-BD) ==> 84 octets
- descripteurs (7) ==> 56 octets
- valeurs initialisées (Z-IDENT + Z-VALIT) ==> 366 octets

soit 539 octets

- 3 en-têtes BC ==> 6 octets

transfert fichier résultat :

- un article ==> 256 octets
- 2 en-têtes BC ==> 4 octets

c-est-à-dire 13 accès au réseau pour transférer 815 caractères.

Conclusion.

Vu que le programme n'est pas complet, dans ce sens qu'aucun traitement d'erreur n'est prévu, l'emploi d'un programme en langage de base ne paraît pas avantageux par rapport à un ensemble de primitives quand il s'agit de créer un article.

Essayons d'extrapoler la situation , si on désire créer plusieurs articles successivement :

- Au niveau primitive, chaque création supplémentaire coûte 4 accès réseau et 592 octets.

- Au niveau du programme en code interne, il faut prendre l'hypothèse que toutes les valeurs sont initialisées. Par article créé on aura besoin d'une instruction d'appel au Processeur Base de Données, d'un descripteur pour une zone Z-VALIT et d'une zone Z_VALIT, soit 276 octets, ce qui occasionne au maximum 2 accès réseaux (il faut 9 accès réseaux pour 8 créations)
  L'avantage d'un programme en langage de base croît dès lors avec le nombre de créations.

5.3.2. La lecture du Ième article dans un fichier séquentiel.

1. Au niveau Primitives.

   les opérations :

   - ouvrir la Base de Données (Z-CODES, Z-IDENT)
   - lire (I-1) articles (Z-CODES)
   - lire le Ième article (Z-CODES, RFIELD)
   - fermer la Base de Données (Z-CODES)

   le trafic :

   protocole : Dmde-Comm + Ack + Fin-Comm + Ack ==> 5 octets

   primitive 1: En-tête PM + Z-CODES + Z-IDENT
   * 2 ==> 256 octets

   primitive 2: En-tête PM + Z-CODES
   * 2
   * (I-1) ==> (I-1)*74 octets

   primitive 3: En-tête PM + Z-CODES
   * 2
   + En-tête SM + RFIELD ==> 334 octets

   primitive 4: En-tête PM + Z-CODES
   * 2 ==> 76 octets

   Ce travail neessite 11 + (I-1)*2 accès au réseau pour la communication de 671 + (I-1)*74 octets.

2. Au niveau langage interne.

   les instructions :

   - ouvrir la Base de Données
     - maj de COP, SREF, PROTECT
     - appel du Processeur BD

   - lire les (I-1) premiers articles
     - maj de COP, COREC, PREF, COGET
     - boucle tant que PREF < i-1
       - appel du Processeur BD
       - maj de PREF

   - lire le Ième article
     - maj de COGET
     - appel du Processeur BD
     - ecriture du résultat dans un fichier
       - ouverture BD (3 maj + appel)
       - ouverture fichier (2 maj + appel)
       - écriture article (2 maj + appel)
       - fermeture fichier (1 maj + appel)
       - fermeture BD (1 maj + appel)

- fermer la Base de Données
    - maj de COP, SREF
    - appel du Processeur BD

les descripteurs :

Z-CODES, Z-IDENT (* 2), RFIELD
COP, PROTECT, COREC, PREF, RREF, COGET
2 variables d'articles

les valeurs :

une zone pour Z-CODES, RFIELD;
deux zones pour Z-IDENT;
deux zones pour variable d'article

le trafic :

protocole : Dmde-Comm + Ack + Fin-Comm + Ack
* 2 (2 communications) ==> 10 octets

transfert fichier code :

- descripteur général et
  4 descripteurs segments ==> 33 octets
- instructions (20 MOVE + 9 CALL-BD
  + 1 test) ==> 280 octets
- descripteurs (12) ==> 96 octets
- valeurs initialisées (2 * Z-IDENT) ==> 180 octets

soit 589 octets

- 3 en-têtes BC ==> 6 octets

transfert fichier résultat :

- un article ==> 256 octets
- 2 en-têtes BC ==> 4 octets

c-est-à-dire 13 accès au réseau pour transférer 865 octets.

Conclusion.

Il apparaît très clairement que le programme en langage de base, même amélioré avec des tests d'erreurs, occasionne moins de trafic que le même travail avec des primitives.

De plus, le travail occasionné au réseau par le programme en langage de base n'est pas fonction de la longueur de la recherche (donnée par la variable I).

5.3.3. La modification ou la suppression du Ième article.

Nous ne détaillerons pas ces deux transactions, car les conséquences sont les mêmes que pour la lecture du Ième article. En effet, la modification ou la suppression d'un article ne peuvent être effectuées que sur l'article courant, c'est-à-dire l'article auquel on vient d'accéder par une lecture.

5.3.4. La sélection de k articles dans un fichier de n articles, sur la valeur d'un item.

1. Au niveau Primitives.

   les opérations :

   La sélection ne pouvant être opérée que par l'utilisateur, cette transaction revient à exécuter les primitives suivantes :

   - ouvrir la Base de Données (Z-CODES, Z-IDENT)
   - lire n articles (Z-CODES, RFIELD)
   - fermer la Base de Données (Z-CODES)

   le trafic :

   protocole : Dmde-Comm + Ack + Fin-Comm + Ack ==> 5 octets

   primitive 1: En-tête PM + Z-CODES + Z-IDENT
   * 2 ==> 256 octets

   primitive 2: En-tête PM + Z-CODES
   * 2
   + En-tête SM + RFIELD
   * n ==> n * 334 octets

   primitive 3: En-tête PM + Z-CODES
   * 2 ==> 76 octets

   On effectuera 8 + n*3 accès au réseau et on transférera 337 + n*334 octets.

2. Au niveau langage interne.

   les instructions :

   - ouvrir la Base de Données (3 maj + appel)
   - ouvrir la Base de Données des fichiers standard (2 maj + appel)
   - ouvrir le fichier résultat (2 maj + appel)
   - boucle de lecture des n articles (test fin boucle)
     - lecture article (4 maj + appel)
     - test valeur de sélection (par ex. 15 instructions)
     - si test positif écriture article (4 maj + appel)
   - fermer le fichier résultat (2 maj + appel)
   - fermer la Base de Données des fichiers standard (2 maj + appel)
   - fermer la Base de Données (2 maj + appel)

les descripteurs :

Z-CODES, Z-IDENT (* 2), RFIELD
COP, PROTECT, COREC, PREF, RREF, COGET
2 variables d'articles

les valeurs :

une zone pour Z-CODES, RFIELD;
deux zones pour Z-IDENT;
deux zones pour variable d'article

le trafic :

protocole : Dmde-Comm + Ack + Fin-Comm + Ack
* 2 (2 communications) ==> 10 octets

transfert fichier code :

- descripteur général et
  4 descripteurs segments ==> 33 octets
- instructions (21 MOVE + 8 CALL-BD
  + 16 tests) ==> 392 octets
- descripteurs (12) ==> 96 octets
- valeurs initialisées (2 * Z-IDENT) ==> 180 octets

soit 701 octets
- 3 en-têtes BC ==> 6 octets

transfert fichier résultat :

- k articles ==> k * 256 octets
- k caractères de séparation ==> k * 4 octets
- moins de 2*k en-têtes BC ==> k * 4 octets

On a besoin de faire maximum 11 + 2*k accès au réseau qui transporteront 717 + k*264 octets.

Conclusion.

A nouveau les chiffres parlent d'eux-mêmes : le programme en langage de base est bien plus économique que l'emploi de primitives. De plus, le trafic engendré par le programme en langage interne est fonction du nombre d'articles sélectionnés tandis que le trafic créé par l'emploi de primitives augmente avec le nombre d'articles du fichier.

Résolvant les équations

- accès : 8 + (3 * n) = 11 + (2 * k)
- octets : 337 + (334 * n) = 717 + (264 * k)

on s'aperçoit que pour atteindre un nombre d'accès équivalent dans les deux cas, le nombre d'articles sélectionnés k doit valoir (3*(n-1))/2 soit être une fois et demi le nombre d'articles du fichier ce qui est absurde.

En ce qui concerne le nombre de caractères, il faut attendre de devoir transférer plus d'une fois le nombre d'articles du fichier ce qui est toujours absurde, avant que les primitives dépassent le langage de base.

5.4. EVALUATION GLOBALE.

Reprenons de façon synthétique les différents résultats que nous avons énoncés.

utilisateur local :

- facilité et efficacité d'utilisation

  (programmeur) IDML >> Lgbs > Primitives
  (non specialiste) IDML >> Primitives > Lgbs

- taille du programme d'application IDML = Lgbs = Primitives

- temps de réponse

  Primitives : exécution de p primitives
  + transfert de 2 * (p+2) messages
  + attente [0 .. (p+1)*T{n-1}]

  Lgbs : chargement et exécution du programme
  + transfert du fichier code
  et du fichier résultat
  + attente [0 .. T{n-1}] + (2*T{n-1})

système IDML :

- taille du système Primitives >> Lgbs >> IDML
- charge du système Primitives >> Lgbs >> IDML

réseau :

- trafic IDML > Lgbs >> Primitives

Quelles conclusions tirer ?

Il apparaît clairement que les intérêts de l'utilisateur local et du réseau vont dans le même sens :

IDML > Langage interne > Primitives,

ce qui s'oppose à la situation du site de traitement (Système IDML partagé) pour qui

IDML < Langage interne < Primitives.

Un élément dont nous n'avons pas beaucoup tenu compte jusqu'ici est le nombre d'utilisateurs simultanément au travail.

Voyons l'influence qu'il a :

- sur le réseau :

  plus les utilisateurs sont nombreux, plus le taux de collision augmente, plus de temps de service se dégrade;

- sur le site de traitement :

  la charge du système croît avec le nombre d'utilisateurs;

- sur un utilisateur :

  le temps de réponse est mis en danger. Il reste à voir à quelle situation il résistera le mieux :

  - travailler au coup à coup, par primitives, avec une attente entre chacune d'elle,
  - lancer le travail et recevoir les résultats, le délai entre les deux étant élevé et inconnu au départ.

Personnellement, nous croyons que l'emploi du langage interne est un compromis qui pourrait satisfaire tous les partenaires, pour autant qu'une découpe plus fine soit effectuée au niveau des tâches.

Les avantages que nous retiendrons :

- la puissance de travail qui peut être effectué;
- le trafic relativement faible sur le réseau;
- l'aspect psychologique lié au problème d'attente.

Les inconvénients principaux :

- le ralentissement général du système partagé;
- la nécessité pour l'utilisateur de savoir programmer en langage interne, ou d'avoir un compilateur approprié (y compris la Base de Données Schémas!).

Néanmoins, il apparaîtra à tous que cette idée ne peut pas être retenue comme LA solution.

Les trois niveaux doivent être disponibles afin de ne léser aucun utilisateur, et ceci au risque de perdre en performance. Charge à l'utilisateur de choisir ce qui lui convient le mieux, en sachant que les performances qu'il peut espérer obtenir sont fonction non seulement du niveau de communication mais aussi des autres utilisateurs, et réciproquement.

CONCLUSION

Nous voici arrivé au terme du projet, non pas parce que celui est achevé, mais parce que le temps qui nous était imparti est écoulé.

Son objectif était d'étudier pratiquement l'extension du système de Manipulation de Bases de Données Hétérogènes IDML mono-utilisateur à un système distribué capable de supporter une Base de Données Répartie sur un réseau de micro-ordinateurs.

Après avoir parcouru la littérature pour dégager les idées principales sur les Bases de Données Réparties, nous avons étudié le système IDML mono-utilisateur, étude qui a fait suite à une implémentation partielle.

A partir de ce système, nous avons cherché à spécifier un système partagé. Et nous l'avons implémenté.

Enfin, nous avons dégagé quelques idées concernant l'extension à un système distribué. Un manque de connaissance et un manque de temps ne nous ont pas permis d'atteindre des conclusions plus précises.

Le travail présenté n'est pas complet :

- vu l'ampleur du concept de Base de Données, certains problèmes n'ont pas été traités (sécurité, contrôle de cohérence, reprise...);
- vu l'importance des programmes à réaliser, et le peu d'outils au départ, l'implémentation n'est pas optimale (découpe en tâche, sauvetage de contexte...)

En ce qui concerne la réalisation pratique, plusieurs points sont à noter.

Tout d'abord, la configuration hardware sur laquelle a été effectuée l'implémentation n'a pas permis d'atteindre les performances attendues en terme de temps d'exécution.

De plus, certains modules n'ont pas mis en application l'entièreté de spécifications.

En particulier, nous n'avons traité que le cas d'un utilisateur local unique (restriction de la configuration) qui, au niveau primitive, travaille uniquement en interactif. La levée de cette restriction permettrait peut-être d'atteindre des performances plus agréables.

Le système IDML est imposant par sa taille et se trouverait sûrement plus à l'aise sur une configuration comportant une mémoire centrale importante et des moyens de stockage secondaire plus performants.

Quant à la résistance du réseau, seule une exploitation plus intense permettra d'en saisir la fiabilité et la valeur réelle.

Ces quelques remarques permettent de voir que le projet est loin d'être conclu. Les possibilités d'extension et d'amélioration sont énormes.

Ce travail constitue un premier pas vers l'objectif de départ. L'élément essentiel à en retenir semble être le suivant : l'objectif ambitieux d'étendre le système IDML à un système distribué sur un réseau de micro-ordinateurs est réaliste. Les problèmes sont nombreux, des solutions restent à trouver, mais les perspectives sont larges. Il y a encore du chemin à parcourir avant de réaliser totalement l'objectif.

B I B L I O G R A P H I E.

Les Bases de Donnèes Rèparties.

[ADIBA] M. ADIBA, C. DELOBEL
" The Problems Of The Cooperation Between Different D.B.M.S "

Architecture and Models in D.B.M.S, G.M. Nyssen (ed.), North Holland Publishing Company, 1977.

[ASCHIM] F. ASCHIM
" Data Base Networks - An Overview "

Management Informatics, Vol 3, No 1, 1974.

[CHUPIN] J.C. CHUPIN
" Rèpartition d'Applications et de Bases de Donnèes sur un Rèseau Gènèral d'Ordinateurs "

Manuscrit de thèse de Doctorat d'Etat, Octobre 1977.

[DAVENPORT] R.A. DAVENPORT
" Distributed or Centralised Data Bases ? "

The Computer Journal, Vol 21, Number 1, Aout 1976.

[DELOBEL] C. DELOBEL
" Les systèmes de Bases de Donnes "

1975

[DEL VECCHIO] B. DEL VECCHIO, P. PENNY
" The PHLOX Project : Three Data Bases Management Systems for Micro-Computers "

ACM / GAL-I-035 I.N.R.I.A.

[GIEN] M. GIEN
" A File Transfer Protocol (FTP) "

I.R.I.A, Rèseau Cyclades, DAT 522, Dècembre 1977.

[HARDGRAVE] W.T. HARDGRAVE
" Distributed Database Technology : An Assessment

Briefings Information & Management 1, North Holland Publishing Company, 1978.

[JOUVE] M. JOUVE, C. PARENT, S. SPACCAPIETRA
" Principes des Systèmes de Gestion de Bases de Données Réparties "

Structure et Base de Données, R.A.I.R.O Informatique/Computer Science, Août 1980.

[LE LANN] G. LE LANN
" Cohérence et Gestion des Accès simultanés dans les Systèmes de Bases de Données Réparties "

I.R.I.A. - Projet SIRIUS
Version française du cours présenté à l'école des Communautés Européennes " Distributed Data Bases " Sheffield (G.B), juillet 1979.

[MILLER] M. MILLER
" A Survey of Distributed Data Bases Management "

Briefings Information & Management 1, North Holland Publishing Company, 1978.

[NAHOURAII] E. NAHOURAII, L.O BROOKS, A.F. CARDENAS
" An Approach to Data Communication Between Different Generalized Data Base Management Systems

Systems for Large Data Bases, P.C Lockemann and E.J Neuhold (eds), North Holland Publishing Company, 1976.

[SCHREIBER] F. SCHREIBER
" Problems and Models in Distributed Database Systems "

Instituto di Elettronica-Politecnico, Milano.

[SPACCAPIETRA] S. SPACCAPIETRA
" Bases de Données Réparties "

Monographie d'informatique de l'AFCET, groupe de travail AFCETT-TTI sur les Bases de Données, animé par S.SPACCAPIETRA, Edition Hommes et Techniques, 1978.

[STONEBRAKER] M. STONEBRAKER
" Concurrency Control and Consistency of Multiple Copies of Data in Distributed INGRES "

I.E.E.E Transactions on Software Engineering, Vol SE-5, No 3, May 1979.

[WILMS] P. WILMS
" Comment disposer et exploiter une Base de Donnèes sur un rèseau ? "

F.B.V.I - F.A.I.B, Congrès 1980.

[ZIEGLER] K. ZIEGLER
" Distributed Data Base, Where Are You ? - (A Tutorial) "

Information Processing I.F.I.P, North Holland Publishing Company, 1977.

Le système IDML .

[DELVAUX] Y. DELVAUX, J.L. HAINAUT
" Système portable de Manipulation de Bases de Donnèes hètèrogènes "

Institut d'Informatique, F.N.D.P. Namur, Mars 1981.

[DELVAUX] Y. DELVAUX
" Projet IDML : Le langage de base "

Institut d'Informatique, F.N.D.P. Namur, Janvier 1980.

[DELVAUX] Y. DELVAUX
" Projet IDML : La reprèsentation des donnèes "

Institut d'Informatique, F.N.D.P. Namur, Janvier 1980.

[HAINAUT] J.L. HAINAUT
" Implèmentation des programmes IDML "

Institut d'Informatique, F.N.D.P. Namur, Juin 1979.

[ROBERT] J.P. ROBERT
" Gènèration automatique d'interfaces d'accès à des Bases de Donnèes Cobol "

Mèmoire de fin d'ètudes, Institut d'Informatique, F.N.D.P. Namur, Septembre 1981.

Le rèseau.

[LEMAL] E. LEMAL
" Contribution à la mise en oeuvre d'un rèseau local "

Mèmoire de fin d'ètudes, Institut d'Informatique, F.N.D.P. Namur, Juin 1982.

[MACCHI] C. MACCHI, J.F. GUILBERT
" Tèlèinformatique : transport et traitement de l'information dans les rèseaux et les systèmes tèlèinformatiques "

Bordas et C.N.E.T - E.N.S.T, Paris, 1979.

[NORTH STAR] NORTH STAR COMPUTERS Inc.
" North Star Computers Product Catalog "

August 1979

Rèalisation pratique.

[JACKSON] M.A. JACKSON
" Principles of Program Design "

Academic Press London, New York, San Francisco, 1975.

[MT MicroSYSTEMS] MT MicroSYSTEMS, Inc.
" Pascal MT+ - User's Guide "

Release 5, Third Edition, Lifeboat Associates, 1981.

[PETITPIERRE] C. PETITPIERRE
"Progammation en temps rèel "

Cours postgrade en Informatique technique, Ecole Polytechnique Fèdèrale de Lausanne, Dècembre 1980.

FACULTES UNIVERSITAIRES NOTRE-DAME DE LA PAIX - NAMUR

INSTITUT D'INFORMATIQUE

DISTRIBUTION DU SYSTEME

IDML

- Interactive Data Manipulation Language -

SUR UN RESEAU DE

MICRO-ORDINATEURS

*** A N N E X E S ***

Mémoire présenté par

Béatrice SCOYER

en vue de l'obtention du titre de
Licencié et Maître en Informatique

Promoteur : J.L. HAINAUT

- Année académique 1981-1982 -

A N N E X E S

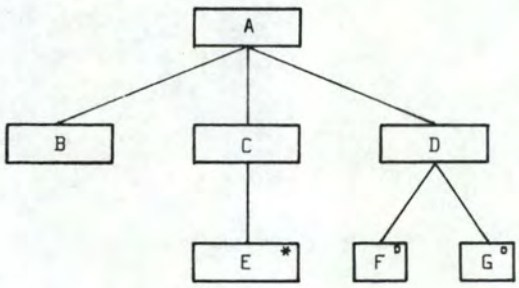
A N N E X E I. LE CONCEPT D'ARBRE PROGRAMMATIQUE.

Le concept d'arbre programmatique, utilisé comme formalisme pour exposer des traitements est issu de la théorie de Jackson.

Le mode de représentation adopté repose sur ce qu'en théorie des graphes on nomme des arborescences. De tels graphes sont fréquemment utilisés et leur compréhension ne nécessite pas de développement particulier.

L'arbre programmatique est un graphe de type arborescent figurant la structuration de l'ensemble des traitements. La recherche de cette structuration se conduit par décomposition successive de la fonction en sous-fonctions, niveau par niveau.

On obtiendra par exemple une structure de ce type :



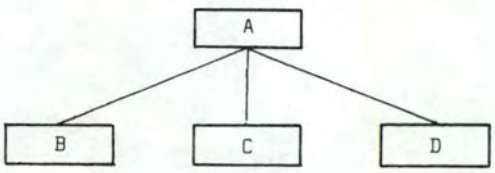
On distingue quatre types d'élèments :

- l'élèment élèmentaire,
- la séquence,
- l'itération,
- la sélection.

1. <u>La séquence</u>.

Une séquence est composée de deux ou plusieurs élèments qui se produisent chacun une fois, dans un ordre pré-défini.

La structure séquentielle est représentée comme suit :

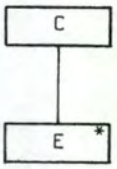


Le traitement A consiste à réaliser les sous-traitements B, puis C, puis D.

2. L'itération.

Une itération est composée d'un élément qui se produit 0 ou plusieurs fois consécutivement.

Elle est représentée comme suit :



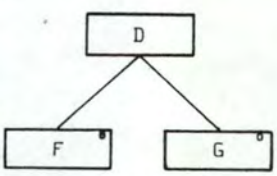
L'étoile dans le cadre de E indique les occurences multiples de E dans C. Il faut noter que C est l'itération et que la multiplicité des occurences de E est un attribut de C.

La terminaison est déterminée par un test de condition.

3. La sélection.

La sélection est constituée de deux ou plusieurs éléments dont un seul se produit.

La structure de sélection est représentée comme suit :



Le rond dans le cadre de F et G indique que D est une sélection parmi ces deux traitements. La sélection est opérée sur base d'une condition.

Le test de condition doit évidemment précéder l'exécution de l'élément.

Un programme est une combinaison de ces structures. Cette combinaison se fait par imbrication entraînant une augmentation du nombre de niveaux dans l'arborescence ou par concaténation à un même niveau.

A un même niveau, l'ordre d'exécution va de la gauche vers la droite. Dans l'arbre programmatique, l'axe horizontal donne l'ordre d'exécution tandis que l'axe vertical correspond au niveau d'imbrication.

A N N E X E II

IMPLEMENTATION DES INTERFACES IDML DE BASE DE DONNEES

1. LE MODELE D'ACCES.

Ce modèle se veut, de part la généralité de ses concepts, un modèle de description de données apte à décrire tout ensemble de données, quel que soit le Système de Gestion de Base de Données ou de Fichier utilisé.

Les concepts du modèle d'accès sont répartis en trois ensembles:

- les structures de données;
- les mécanismes d'accès;
- les primitives.

1.1. LES STRUCTURES DE DONNEES.

1. La Base de Données.

   C'est la collection des articles d'un ensemble de fichiers; elle contient toujours un article particulier (virtuel ou non) qui est son point d'entrée et qui porte le nom de cette Base de Données.

2. Le fichier.

   Un fichier est une collection dynamique d'articles; un article appartient toujours à un fichier.

3. Le type d'article.

   Il décrit les propriétés générales des articles qui appartiennent à ce type.

4. L'item.

   L'item est un type d'information défini par un ensemble de valeurs. Un item est associé à au moins un type d'article. Il porte un nom qui l'identifie parmi tous les items de la Base de Données.

   Un item peut être élémentaire ou décomposable : la valeur d'un item élémentaire est atomique du point de vue de sa signification; la valeur d'un item décomposable est une liste de valeurs significatives; chacune de ces dernières appartient à un item qui est dit composant de l'item décomposable. Un composant peut être lui-même décomposable.

Un item peut être simple ou répétitif : il est dit simple si à chaque article n'est jamais associée plus d'une valeur de cet item; il est répétitif dans le cas contraire.

Un item peut être obligatoire ou facultatif : il est obligatoire pour un type d'article si a ≙ chaque article de ce type est associée au moins une valeur de cet item; il est facultatif sil est permis de n'associer aucune valeur de cet item à un article de ce type.

5. L'article.

Unité d'information enregistrée, il peut faire l'objet d'une demande d'accès, de création, de modification ou de suppression. Tous les articles sont distincts et chacun appartient à un seul type d'article.

6. La valeur d'item.

Donnée manipulable par un programme, ses propriétés sont décrites par l'item auquel elle est associée.

7. Notion d'identifiant.

Un identifiant est un item (ou une liste d'items) d'un type d'article tel qu'il n'existe pas, dans le référentiel précisé (par exemple la Base de Données ou le fichier), plus d'un article qui soit associé à une même valeur de cet item (ou des items de cette liste).

8. Clé d'accès.

Une clé d'accès est un item (ou une liste d'items) d'un type d'article tel qu'il existe un mécanisme qui permette d'accéder successivement aux articles auxquels est associée une valeur déterminée de cette clé.

Une clé d'accès peut être identifiante ou non.

Une clé d'accès est caractérisée par le référentiel dans lequel elle porte ses effets.

9. Le chemin d'accès.

Le chemin d'accès est un mécanisme qui associe un article dit origine à 0, 1 ou plusieurs articles dits cibles, d'une manière telle qu'il soit possible, à partir de son article origine, d'accéder successivement aux différents articles cibles ainsi associés.

10. Type de chemin d'accès.

Tout chemin appartient à un et un seul type qui en définit les propriétés générales. Un type de chemin est caractérisé par les types d'articles auxquels doivent appartenir d'une part les articles origines, et d'autre part les articles cibles de ses chemins.

11. L'article Système.

Toute Base de Données contient un et un seul article d'un type particulier qui porte le nom SYSTEME.

Cet article constitue un point d'entrée privilégié dans la Base de Données; il peut être origine de chemins d'accès.

12. Chemin d'accès implicite.

Certains systèmes offrent des primitives d'accès séquentiel à des articles, alors qu'aucun chemin d'accès n'a été déclaré pour ces derniers. Tel serait le cas de l'accès:

- à tous les articles de la Base de Données;
- à tous les articles d'un type dans la Base de Données;
- à tous les articles d'un fichier;
- à tous les articles d'un type dans un fichier.

On peut décrire ces possibilités sous forme d'un chemin d'accès (implicite) ayant pour origine l'article Système.

13. Ordre des articles cibles dans un chemin d'accès.

L'accès aux articles cibles offert par le mécanisme de chemin d'accès étant de nature essentiellement séquentielle (accès aux cibles successives), il convient, dans certains cas, de pouvoir spécifier une règle qui définit l'ordre des articles cibles dans la séquence d'un chemin.

Quatre classes d'ordre peuvent être envisagées:

- quelconque;
- lié au moment de l'insertion: chronologique ou antichronologique;
- trié selon les valeurs d'une clé de tri (croissantes ou décroissantes);
- dynamique: laissé à l'initiative de l'utilisateur.

1.2. LES MECANISMES D'ACCES.

L'accès est pour un programme la mise à disposition d'un objet de la Base de Données. Cette opération n'est possible qu'à travers les mécanismes d'accès que voici:

- l'accès à une Base de Données;
- l'accès à un fichier;
- l'accès aux articles d'une Base de Données, d'un fichier ou d'un type déterminé;
- l'accès aux valeurs d'items d'un article;
- l'accès aux articles auxquels est associée une valeur déterminée d'un item (clé d'accès);
- l'accès à des articles à partir d'un article (chemin d'accès inter-article).

1.3. LES PRIMITIVES.

Elles correspondent aux opérations élémentaires qu'il est possible d'effectuer sur les objets d'une Base de Données. Trois classes de primitives peuvent être distinguées:

- les primitives d'accès qui mettent en oeuvre les mécanismes d'accès;
- les primitives de modification qui font évoluer la Base de Données;
- les primitives de contrôle d'environnement qui permettent la création de super-primitives, l'établissement de points de reprise et la maîtrise de la sécurité et de la concurrence.

2. LE SYSTEME DE GESTION DE FICHIERS PASCAL MT+.

2.1. DEFINITIONS.

Un FICHIER est une collection de donnèes organisèes en enregistrements logiques de même taille.

La taille et l'organisation des donnèes est controlèe par l'utilisateur dans son programme.

Deux modes d'accès aux donnèes sont possibles :

1. l'accès sèquentiel;
2. l'accès direct sur base d'une clè offerte par le système.

Le type d'un fichier dèfinit la taille et le format d'un èlèment individuel du fichier, la plus petite unitè accesible d'un fichier.

Le type d'un fichier induit le TYPE D'ARTICLE prèsent dans le fichier.

Plusieurs types d'articles sont disponibles et offrent des possibilitès diverses.

1. l'èlèment est de type scalaire standard : un boolèen, un entier, un caractère ou un rèel;

2. l'èlèment est de type structurè : une chaîne de caractères, un tableau ou un record;

3. l'èlèment est du texte : le type de fichier TEXTE est utilisè pour les fichiers Ascii. Le fichier de texte est similaire au fichier de caractères exceptè qu'il est subdivisè en lignes, que les nombres qu'on y ècrits sont convertis en ASCII et peuvent être formattès et que les nombres qu'on en lit sont tranformès en binaire.

   Une ligne est une sèquence de caractères terminèe par un symbole de fin de ligne (le caractère Ascii <RETURN> suivi du caractère Ascii <LINE FEED>).

4. le type de l'èlèment n'est pas prècisè: le fichier sans type est utilisè pour des Entrèes/Sorties rapides de bloc de caractères, sans se prèoccuper du type des donnèes contenues logiquement dans le fichier.

2.2. LES OPERATIONS DE BASE SUR UN FICHIER (accès séquentiel).

Les opérations fondamentales sur un fichier sont au nombre de quatre: la création d'un fichier, l'ouverture du fichier pour la consultation, la création et la lecture d'un enregistrement.

1. Créer un fichier - procédure REWRITE.

   Cette procédure a pour but de générer un fichier sur disque pour y recevoir des articles: si le fichier existe déja, il est effacé; s'il n'existe pas encore un nouveau fichier est créé.
   Le nom du fichier sur disque se trouve dans un bloc d'informations associé au fichier en question. Ce nom a du être préalablement attribué par l'utilisateur, et ce grâce à la procédure ASSIGN qui permet d'établir la correspondance entre le nom du fichier dans le programme et le nom du fichier sur disque.

   Le fichier est prêt à recevoir des articles. Ces articles seront stockés séquentiellement (c'est-à-dire dans l'ordre de leur création) à partir de la première position du fichier.

2. Création d'un article dans le fichier - procédure PUT.

   A chaque fichier est associée lors de son ouverture une zone tampon qui a le type spécifié de l'article composant le fichier.

   Les valeurs de l'article que l'on veut créer doivent avoir été chargées dans cette zone. La procédure PUT a pour effet de transférer le contenu de cette variable tampon dans le fichier. Ce nouvel article est ajouté à la fin du fichier (organisation séquentielle).

3. Ouverture d'un fichier pour la consultation - procédure RESET.

   Cette procédure permet l'accès en lecture aux articles du fichier. Le fichier est positionné au début et la valeur de la première composante est transférée dans la zone tampon.

   A nouveau, le nom du fichier se trouve dans un bloc d'informations lié au fichier. Ce nom a été garni par la procédure ASSIGN.
   Si le fichier qu'on désire consulter n'existe pas, l'erreur est signalée.Par contre, si le fichier existe mais est vide, l'ouverture se fera correctement. L'absence d'articles sera signalée lors de la première lecture.

4. Lecture d'un article du fichier - procédure GET

   L'appel à cette procédure a pour effet de lire une composante du fichier, c'est-à-dire de se positionner au début de la composante suivante et d'amener dans la zone tampon une copie de cette composante si elle existe. Si cete composante n'existe pas, un indicateur de fin de fichier est positionné.

Il faut encore ajouter quelques opérations pour pouvoir cloturer le travail sur un fichier.

- procédure CLOSE : tout fichier précédemment ouvert doit être fermé.

- procédure CLOSEDEL : le fichier ouvert est fermé et détruit. Cette opération est utilisée avec des fichiers temporaires.

- procédure PURGE : le fichier est détruit. Cette procédure est a appeler si le fichier est fermé.

2.3. LES OPERATIONS PARTICULIERES SUR UN FICHIER DE TYPE TEXTE.

On appelle fichier de texte un fichier dont les composantes sont des caractères. Il est structuré en lignes; la fin de ligne étant indiquée par un symbole particulier.
On peut écrire et lire dans ces fichiers de texte non seulement des caractères mais aussi des entiers, des réels et des booléens.

Les opérations de base sur les fichiers quelconques sont évidemment disponibles sur les fichiers de texte : Rewrite, Reset, Put et Get.

En plus de ces opérations, les commandes Write, Writeln, Read et Readln sont permises sur les fichiers de textes.

1. Création d'un article - procédure WRITE

   Cette opération combine l'accès à la zone tampon et l'opération Put. L'accès au tampon consiste à y ranger les valeurs de l'article à créer.

2. Ecriture d'un symbole de fin de ligne - procédure WRITELN

   Comme le marqueur de fin de ligne est un symbole particulier, il est difficile de l'écrire directement à partir du programme. Le but de cette procédure est justement de permettre à l'utilisateur d'ajouter ce marqueur au fichier afin de structurer son texte en lignes.

3. Lecture d'un article - procédure READ

   Cette procédure facilite la programmation de l'accès au tampon suivi d'une commande Get. L'accès au tampon a pour objet de sauvegarder dans une variable de l'utilisateur les valeurs actuellement dans cette zone tampon.

4. Lecture d'un marqueur de fin de ligne - procédure READLN

   L'opération READLN place l'indicateur de position du fichier sur le caractère suivant le prochain marqueur rencontré.
   Par contre, la lecture du marqueur avec la commande Get amène un blanc dans le fichier.

2.4. OPERATIONS D'ACCES DIRECT SUR UN FICHIER.

Un fichier à accès direct est un fichier Pascal avec un type. Ce fichier est accédé via deux procédures particulières. Ainsi, n'importe quel fichier peut être considéré comme un fichier relatif: il n'y a pas de syntaxe spéciale à utiliser quand on désire accéder de façon directe plutôt que séquentielle aux articles d'un fichier.

Ceci est possible grâce au mécanisme qu'utilise le système pour stocker ses enregistrements. Lors de la création d'un article, une valeur de clé interne au fichier est attribuée à cet article. La numérotation des éléments commence à zéro et est relative au début du fichier.

Cette clé interne permet de retrouver un article dans le fichier. Le plus grand numéro attribué constitue la fin du fichier. De la valeur zéro à cette valeur de fin de fichier, toute les valeurs intermédiaires ont été logiquement attribuées. Si un article d'une de ces clés intermédiaire n'a pas été réellement créé, les valeurs contenues dans le fichier pour cet article sont indéfinies. Le système ne fera pas de distinction entre un article réellement et un article logiquement écrit. Cette gestion doit être effectuée par l'utilisateur.

Les opérations que l'on peut effectuer sont les suivantes : création d'un fichier, ouverture d'un fichier en consultation/modification, écriture d'un article et lecture d'un article.

1. Un fichier est créé par l'ordre REWRITE.

2. Un fichier existant peut être lu, modifié et étendu après la commande RESET (ce qui n'est pas le cas en accès séquentiel où l'ordre Reset n'autorise qu'un accès en lecture et où il n'existe pas directement d'opération de modification ou d'extension d'un fichier).

3. Ecriture d'un article - procédure SEEKWRITE

   Cette opération a pour effet de tranférer le contenu de la zone tampon à l'emplacement indiqué par le numéro de record donné par l'utilisateur. Si le numéro correspond à un emplacement déja réservé dans le fichier, les valeurs qui s'y trouvent sont écrasées par les nouvelles valeurs.

4. Lecture d'un article - procédure SEEKREAD

   Cette procédure copie dans la zone tampon le contenu de l'enregistrement du fichier dont le numéro est spécifié par l'utilisateur. Si le numéro d'article dépasse la fin du fichier, l'erreur est signalée.

2.5. OPERATIONS SUR LES FICHIERS SANS TYPE.

Le Pascal MT+ permet d'utiliser des fichiers dont le type des éléments n'est pas précisé.

Les opérations sont similaires aux opérations d'accès direct : une opération Rewrite initialise un nouveau fichier, après une opération Reset, le fichier est accessible en lecture et en modification.

Les échanges entre le programme et le fichier se font par l'intermédiaire de tableaux d'octets. La taille de ce tableau peut varier de 128 à 2048 (multiple entier de 128).

L'accès peut être direct (on indique le numéro relatif du bloc) ou séquentiel (le numéro relatif du bloc est -1).

La procédure d'écriture est BLOCKWRITE et la procédure de lecture est BLOCKREAD.

Il y a encore deux autres opérations qui permettent un accès séquentiel rapide au niveau du byte. Le fichier est composé de tableaux de [128..4096] caractères.

Ces opérations sont réalisées par les fonctions WNB et GNB.

3. LES INTERFACES BASES DE DONNEES DANS LE SYSTEME IDML.

3.1. LE CONTEXTE.

Le mot interface désigne un programme, spécifique à une Base de Données chargé de réaliser toutes les requêtes émises par des utilisateurs, qui concernent cette Base de Données.

Une requête consiste en une demande d'exécution d'une des primitives définies dans le modèle d'accès.

Le rôle de l'interface est de présenter à l'utilisateur un langage de manipulation de données unique, quel que soit le SGBD gérant la Base de Données.
L'interface exécute une requête (primitive du modèle d'accès) au moyen des ordres du langage de manipulation de données du SGBD gérant réellement la Base de Données.

L'interface est autonome : aucune contrainte n'est imposée sur l'environnement dans lequel elle est appelée à être utilisée : que l'on se trouve dans un contexte "multi Bases de Données" ou "mono Base de Données", l'interface et son utilisation sont inchangées.
Elle doit être capable de prendre en charge la gestion simultanée de plusieurs utilisateurs.

L'interface constitue le moyen de matérialiser l'avantage offert par un modèle unique de description des SGBD tel que le modèle d'accès.

Tout utilisateur familiarisé au modèle d'accès, sans connaître les particularités de tel ou tel SGBD, pourra, s'il dispose de la description en termes du modèle d'accès d'une Bases de Données y accéder au travers d'un interface

3.2. SPECIFICATION GENERALE DE L'INTERFACE.

Une interface est invoquée par un appel; un appel correspond à une demande d'exécution d'une primitive.
Les paramètres et zones de transmission d'informations sont transmis au moment de l'appel, à l'interface.

Ces spécifications sont issues du travail de J.P. ROBERT.

3.2.1. Notions particulières

Notions de référence à un objet.

La référence à un objet est un identifiant associé automatiquement par l'interface, à tout objet auquel on a accédé avec succès et qui permet de le désigner lors de toute demande ultérieure le concernant.

Les objets susceptibles de se voir attribuer une référence sont les Bases de Données, les fichiers et les articles.

La référence Base de Données n'est pas attribuée par l'interface. Cette référence, qui permet d'identifier une Base de Données parmi d'autres, ne joue pleinement son rôle que dans un contexte multi-Bases de Données.

Durée de vie d'une référence à un objet.

Toute référence disparaît avec le processus durant l'exécution duquel elle a été attribuée.

Pendant l'exécution du processus, une référence attribuée ne le sera pas une seconde fois à un autre objet.

Utilisation de la référence à un article d'un fichier.

Les possibilités d'accéder à un objet sur base de sa référence, vont dépendre du SGBD-cible et des options prises lors de la spécification de l'interface.

Ainsi, l'option implémentée permet de réaliser l'accès par référence à un article du fichier, pour autant que ce soit le dernier article de ce fichier auquel il ait fait accès.

En d'autres termes, il faut que cette référence corresponde à la référence courante dans le fichier pour l'utilisateur en question.

Une ambigüité se produit lorsqu'on procède à des suppressions d'articles, sur base de la référence d'article : l'article supprimé, sa référence n'en est plus une.

Si l'on veut accéder à l'article suivant celui supprimé, on considérera que la référence de l'article supprimé peut encore être utilisée comme référence de l'article précédent.

Notion de code de types d'objets.

Rappelons les différents types d'objets que nous avons retenus dans le modèle d'accès : la Base de Données, le fichier, le type d'article, le type de chemin, l'item et la clé d'accès.

Deux possibilités de désignation ont été choisies, soit le nom complet, soit un code.

Les deux possibilités ont été utilisées dans l'interface : la Base de Données et les fichiers seront désignés par leur nom, tandis que les types d'articles, les types de chemins, les items et les clés d'accès le seront par un code.

Ces noms et codes sont connus de l'interface, qui pourra ainsi faire les vérifications nécessaires.

3.2.2. Les paramètres d'appel.

En toute généralité, cinq types particuliers de paramètres sont distingués :

- les paramètres précisant l'opération à effectuer et les objets sur lesquels elle s'applique :
  Z-CODES

- les paramètres permettant le transfert de valeurs d'identifiants ou de valeurs d'objets de l'utilisateur vers l'interface :
  Z-IDENT / Z-VALUES

- des paramètres pour la transmission d'une liste de codes d'items:
  Z-ITEMS

- des paramètres pour la réception de valeurs d'items
  RFIELD

- des paramètres concernant les chemins d'accès :
  Z-SETS

La taille des paramètres est standard, quelle que soit l'implémentation :

- Z-CODES : 30 octets
- Z-IDENT / Z-VALUES : 256 octets
- Z-ITEMS : 32 octets
- RFIELD : 256 octets
- Z-SETS : 164 octets

1. Les arguments Z-CODES

   - COP code opération

     un code opération a été attribué à chaque primitive, lors de tout appel, COP doit contenir l'un de ces codes. Si tel n'est pas le cas, aucune primitive ne sera exécutée.

   - SREF référence du schéma de la Base de Données

     cette référence qui identifie la Base de Données est fournie à l'interface lors du premier appel (ouverture de la Base de Données); SREF devra contenir cette même référence lors de tous les appels successifs, sinon aucune action ne sera faite par l'interface.

   - COREC code de type d'articles

     lors de la demande d'exécution d'une primitive qui requiert parmi ses paramètres une identification de type d'articles, COREC doit contenir le code numérique qui identifie type d'article en question.

- RETCODE code de retour

  RETCODE contient, après l'exécution d'une requête par l'interface, le diagnostic de cette exécution (valeur 0 signifiant une exécution normale; toute autre valeur signifiant la détection d'une anomalie).

- PROTECT code de protection

  il existe deux interprétations de ce paramètre.

  - en ouverture de la Base de Données :
    les valeurs reconnues sont 1 et 2:

    - 1 signifie mode d'ouverture normal des fichiers: on ne peut accéder à aucun article de la Base de Données sans avoir ouvert le fichier qui le contient.

    - 2 signifie mode automatique d'ouverture des fichiers: les accès à des articles sont permis sans ouverture préalable du fichier. Cette possibilité permet des accès rapides occasionnels avec un protocole simplifié; en fait, elle permet d'ignorer la notion de fichier.

  - en ouverture de fichier :
    PROTECT permet de préciser la protection désirée (aucune, protégée, exclusive) par un utilisateur sur un fichier lorsqu'il ouvre celui-çi. On y associe souvent la notion d'intention de travail (consultation, mise-à-jour).

    Les valeurs de PROTECT peuvent être dans ce cas:

    2. consultation
    3. mise à jour
    4. consultation protégée
    5. mise à jour protégée
    6. consultation exclusive
    7. mise à jour exclusive

- COGET code d'accès aux valeurs d'items d'un article

  COGET permet, lors d'un accès à un article, de signaler à l'interface comment on désire accéder aux valeurs d'item de cet article. Trois valeurs sont prévues :

  0. pas d'accès aux valeurs d'items
  1. accès à toutes les valeurs d'items
  2. accès aux valeurs d'items dont les codes des items se trouvent dans Z-ITEMS.

- CONTRL contrôle d'article

  CONTRL permet de préciser le contrôle que l'interface doit exercer sur l'article auquel on accède. Une étude ultérieure de ces mécanismes généralisée à d'autres modèles de données permettra de préciser davantage le rôle de ce paramètre.

- RFIL référence de fichier

  RFIL est garni par l'interface lors de l'ouverture du fichier; toute requête ultérieure concernant ce fichier devra fournir cette référence.

- RREF référence d'article

  Après un accès réussi à un article, RREF contient la référence de cet article; RREF doit être garni de cette référence pour l'appel de certaines primitives.

- PREF référence de l'article précédent

  PREF est utilisé pour l'accès aux articles; il doit contenir la référence du dernier article auquel on a accédé. Si PREF = 0, on accédera au premier article (d'un type ou d'un fichier suivant la primitive).

- COKEY code clé d'accès

  COKEY doit contenir le code d'une clé d'accès lors de l'appel des primitives requérant une clé d'accès; lorsque RFIL et COKEY figurent simultanément dans une liste d'appel, COKEY ne peut contenir que le code d'une clé définie sur le fichier dont la référence est contenue dans RFIL.

- OPERAT opérateur

  Cet opérateur permet, associé à Z-CLE, de préciser une condition sur la clé de l'article auquel on accède par clé. La condition est "clé article accédé OPERAT Z-CLE".
  Les valeurs reconnues pour OPERAT sont:

  0. pas de condition
  1. =
  2. >
  3. not <

- COMOD en réserve

- COSET code d'un type de chemin

  Il contient le code d'un type de chemin d'accès inter-articles, lors de l'appel des primitives qui en exigent un.

- OREF référence d'article origine d'un chemin (ou cible de chemin)

  Il contient la référence de l'article origine d'un chemin d'accès lors de l'appel des primitives faisant intervenir des mécanismes d'accès inter-articles.

- TYP en réserve

2. Les arguments Z-IDENT / Z-VALUES

- **SSNAME nom de la Base de Données**

  ce nom est à fournir lors de la demande d'ouverture de la Base de Données.

- **PSW mot de passe**

  mot de passe à fournir lors de la demande d'ouverture de la Base de Données. S'il ne correspond pas à celui défini pour la Base de Données gérée par l'interface, l'accès ne sera pas autorisé.

- **FILNAME nom de fichier**

  nom à fournir lors de la demande d'ouverture d'un fichier; ce nom doit être celui de l'un des fichiers appartenant à la Base de Données gérée par l'interface.

- **Z-VALIT valeur des items d'un article**

  destiné à contenir les valeurs d'items des articles lors de la création d'un article, ou de modification de valeur d'items d'un article.

- **Z-CLE valeur de la clé d'accès**

  Lors de l'accès par clé, la valeur de la clé dont le code est contenu dans COKEY, figurera dans ce paramètre.

3. L'argument Z-ITEM

Il contient une liste de codes d'items.
Cette liste est destinée à définir le contenu de Z-VALIT et RFIELD. Les codes d'items correspondent successivement aux items dont la valeur est présentée dans Z-VALIT ou dans RFIELD.

4. L'argument RFIELD

RFIELD est destiné à recevoir les valeurs des items de l'article auquel vient d'accéder l'utilisateur.

5. Les arguments Z-SETS

- **STKREF référence de stockage**

  Elle est destinée à recevoir le paramètre de stockage physique, nécessité par certains Systèmes de Gestion de Bases de Données (exemple en Codasyl: location mode direct).

- **SETLST liste de codes de types de chemin d'accès**

  Cette liste contient des codes permettant l'identification des types de chemin auxquels appartiennent les chemins auxquels sont associés les articles dont les références sont dans CURLST, utilisée lors de la création d'un article.

- **CURLST liste de références d'articles**

  Les références contenues dans CURLST sont celles des derniers articles associés à des chemins, auxquels on ait accédé. Les codes de ces chemins sont dans SETLST; CURLST est utilisé pour des créations d'articles.

3.2.3. Les primitives.

Pour chacune des primitives, nous donnons les paramètres d'entrée et de sortie, et la fonction. Nous indiquons les diagnostics d'erreurs susceptibles d'être transmis à l'utilisateur par l'interface (RETCODE).

L'ordre dans lequel ces diagnostics sont renseignés indique pour chaque primitive l'ordre dans lequel les vérifications sont faites par l'interface. Un seul de ces diagnostics étant transmis par appel à l'interface, si plusieurs sont possibles, celui qui figure le plus tôt dans la liste, sera seul transmis

Certains diagnostics, établis à l'entrée de l'interface, sont communs à toutes les primitives.

- "SREF ne contient pas la référence de la Base de Données" qui a été fournie à l'interface lors de la première ouverture de la Base de Données. Ce diagnostic est valable pour toute les primitives sauf l'ouverture de la Base de Données.

- "opération non définie" si COP ne contient pas le code d'une primitive.

- "opération non implémentée" si COP contient le code d'une primitive existante mais non implémentée par l'interface appelée.

- "la Base de Données n'est pas ouverte" si un appel autre que l'ouverture de la Base de Données est transmis alors que la Base de Données est fermée.

Les primitives d'accès

1. Accès à la Base de Données.

   * Ouverture de la Base de Données.

     entrée : COP = 11, SSNAME, PSW, PROTECT, SREF

     sortie : SREF, RETCODE

     fonction :

     - initialiser les accès à une Base de Données. Cette primitive doit être exécutée avec succès, au moins une fois avant toute autre primitive.

     - vérifier l'existence de la Base de Données dont le nom est contenu dans SSNAME.

     - vérifier l'exactitude du mot de passe contenu dans PSW.

     - si la Base de Données est fermée au moment de l'appel, la référence Base de Données contenue dans SREF sera mémorisée et devra être contenue dans SREF lors de tous

les appels ultérieurs, autre qu'une demande d'ouverture.

- si la Base de Données est déjà ouverte (ouvertures multiples), SREF sera garni par l'interface avec la référence fournie (dans SREF) lors de la première demande d'ouverture, quelle que soit la valeur d'entrée de SREF.

- mémoriser le mode d'accès aux articles des fichiers contenu dans PROTECT.
  PROTECT = 0 : mode normal; ouverture obligatoire des fichiers avant tout accès à l'un de leurs articles.
  PROTECT = 1 : mode automatique; ouverture non obligatoire des fichiers avant tout accès à l'un de leurs articles (la notion de fichier est ignorée).

- diagnostics d'erreurs : (transmis dans RETCODE)

  "BD inconnue" si SSNAME ne contient pas le nom de la Base de Données gérée par l'interface.

  "mot de passe incorrect" si PSW ne contient pas le mot de passe de la Base de Données gérée par l'interface.

  "PROTECT incorrect" si PROTECT est < 1 ou > 2.

* Fermeture de la Base de Données.

  entrée : COP = 12, SREF.

  sortie : RETCODE.

  fonction :

  - clôturer les accès à une Base de Données gérées par l'interface et dont la référence est dans SREF.

  - rem : si plusieurs ouvertures de Base de Données ont été exécutées, celle-ci ne sera effectivement fermée que si le nombre de fermetures est égal au nombre d'ouvertures; de plus, au moment de la fermeture effective de la Base de Données, les fichiers encore ouverts seront fermés automatiquement par l'interface.

  - diagnostics d'erreurs (transmis dans RETCODE) :

    "BD non ouverte" si, au moment de l'appel, la Base de Données n'est pas ouverte.

2. Accès aux fichiers.

* Ouverture des fichiers.

  entrée : COP = 21 / 22, SREF, FILNAME, PROTECT.

  sortie : RETCODE, RFIL.

fonction :

- initialiser l'accès à un (COP = 22) ou à tous (COP = 21) les fichiers de la Base de Données dont la référence se trouve dans SREF.

- La protection établie sur le fichier pendant le travail de l'utilisteur sera fonction de la valeur contenue dans PROTECT (cfr. la définition de PROTECT)
  PROTECT = 2 : consultation sans protection
  PROTECT = 3 : mise à jour sans protection
  PROTECT = 4 : consultation protégée
  PROTECT = 5 : mise à jour protégée
  PROTECT = 6 : consultation exclusive
  PROTECT = 7 : mise à jour exclusive.

- si COP = 22 (ouverture d'un fichier), FILNAME doit contenir le nom du fichier que l'on veut ouvrir; après l'exécution de la primitive, RFIL contiendra la référence attribuée au fichier par l'interface.

- si plusieurs ouvertures sur le même fichier sont demandées, les droits restent ceux établis lors de la première ouverture.

- diagnostics d'erreurs (transmis dans RETCODE) :

  "FILNAME invalide" si FILNAME ne contient pas le nom d'un fichier de la Base de Données.

  "PROTECT incorrect" si PROTECT < 1 ou > 7.

  "fichier non disponible" si le fichier, dont le nom est dans FILNAME, n'est pas accessible ou n'existe pas.

  "mode ouverture invalide" si le fichier a été, lors de sa première ouverture, ouvert en consultation (en mise à jour), on cherche à l'ouvrir lors d'un appel ultérieur, en mise à jour (en consultation).

* Fermeture des fichiers.

entrée : COP = 23 / 24, SREF, RFIL.

sortie : RETCODE.

fonction :

- clôturer l'accès à un (COP = 24) ou à tous (COP = 23) les fichiers de la Base de Données dont la référence est dans SREF.

- si on désire la fermeture d'un seul fichier (COP = 24), RFIL doit contenir la référence de ce fichier.

- une fermeture doit toujours avoir été précédée par au moins une ouverture; cependant COP = 24 sera interprété comme étant une demande de fermeture de tous les fichiers ouverts.

- la fermeture d'un fichier n'est effective que si le nombre de fermetures est égal au nombre d'ouvertures demandées pour ce fichier.

- si la fermeture est effective le(s) fichier(s) est(sont) devenu(s) inacessible(s) pour l'utilisateur.

- diagnostics d'erreurs (transmis dans RETCODE) :

  "RFIL invalide" si RFIL ne contient pas une référence attribuée à un fichier de la Base de Données.

  "fichier non ouvert" si RFIL ne contient pas une référence de fichier ouvert ou si aucun fichier n'était ouvert.

3. Accès aux articles.

Les primitives suivantes permettent d'accéder à des articles et à leurs valeurs d'items. On peut distinguer quatre types parmi ces primitives :

- accès aux articles d'un type,
- accès aux articles d'un fichier,
- accès aux articles cibles d'un chemin,
- accès aux articles par leur référence.

Les trois premiers types sont l'objet d'une subdivision supplémentaire en accès séquentiel et accès par clé.

PREF doit contenir la référence du dernier article auquel on ait accédé. Dans le cas où cet article a été supprimé, on considère que sa référence lui survit jusqu'à l'accès suivant : PREF doit contenir la référence de l'article supprimé lors de l'accès à l'article suivant.

* Accès séquentiel aux articles d'un type donné.

<u>entrée</u> : COP = 31, SREF, COREC, COGET, PREF, COKEY.

<u>sortie</u> : RETCODE, RREF, RFIL, RFIELD.

<u>fonction</u> :

- fournir (dans RREF) la référence de l'article dont le type est donné dans COREC et qui suit celui dont la référence est contenue dans PREF.

- garnir RFIL de la référence du fichier qui contient les articles du type donné.

- si COGET = 1, les valeurs d'item de l'article seront placées dans RFIELD.

- si PREF = 0, l'accès concerna le premier article du type donné.

- si aucune clé d'accès n'est définie sur le fichier, l'ordre de l'accès séquentiel aux articles d'un type sera celui de leur création : du premier au dernier.
  Si une clé d'accès a été définie sur le fichier, on accède aux articles dans l'ordre des valeurs croissants de la clé.
  Si plusieurs clés sont définies, COKEY contiendra le code de la clé dont les valeurs croissantes détermineront l'ordre de l'accès séquentiel.

- diagnostics d'erreurs (transmis dans RETCODE) :

  "COREC invalide" si COREC ne contient pas le code d'un type d'article de la Base de Données.

  "COKEY invalide" si COKEY ne contient pas le code d'une clé d'accès définie sur le fichier contenant le type d'article dont le code est dans COREC.

  "COGET invalide" si COGET > 1.

  "fichier non ouvert" si, lors d'une tentative d'accès à un article d'un fichier fermé si l'ouverture de la Base de Données s'est faite en mode normal.

  "pas d'article trouvé" si le dernier article auquel on a déjà accédé était le dernier du type considéré dans le fichier.

* Accès par clé aux articles d'un type donné.

entrée : COP = 32, SREF, COREC, COGET, PREF, COKEY, OPERAT, Z-CLE

sortie : RETCODE, RREF, RFIL, RFIELD.

fonction :

- fournir dans RREF la référence de l'article dont le type est donné dans COREC et qui vérifie la condition "COKEY OPERAT Z-CLE".

- PREF doit être garni avec la référence du dernier article de ce type auquel on a accédé;
  si PREF = 0, on accède au premier, sinon l'article auquel on accède sera le suivant de celui dont PREF contient la référence.

- fournir dans RFIL la référence du fichier qui contient cet article.

- si COGET = 1, les valeurs d'item de l'article seront retournées dans RFIELD.

- diagnostics d'erreurs (transmis dans RETCODE) :
  les mêmes que ceux rencontrés dans l'accès séquentiel, avec en plus :

  "pas d'article trouvé" si aucun article dont la valeur de clé ne satisfait la condition donnée, n'a pu être trouvé.

* Accès séquentiel aux articles d'un fichier.

entrée : COP = 33, SREF, RFIL, COREC, PREF, COGET, COKEY.

sortie : RETCODE, RREF, RFIELD, COREC.

fonction :

- fournir dans RREF la référence de l'article contenu dans le fichier dont la référence se trouve dans RFIL; si COREC est différent de 0, cfr.: accès séquentiel aux articles d'un type.

- la séquence dans laquelle on accède aux articles sera celle définie pour l'accès séquentiel aux articles d'un type.

- PREF doit contenir la référence du dernier article de ce fichier ou de ce type auquel on a accédé; si PREF = 0, on accèdera au premier.

- si COGET = 1, les valeurs d'item de l'article seront transférées dans RFIELD.

- si COREC = 0, si un seul type d'article est contenu dans le fichier ou si une condition d'identification de type d'article a été spécifiée, COREC contiendra après l'exécution de la primitive le code du type de l'article auquel on a accédé.

- diagnostics d'erreurs (transmis dans RETCODE) :

  les mêmes que ceux définis pour l'accès séquentiel aux articles d'un type.

* Accès par clé aux articles d'un fichier.

entrée : COP = 34, SREF, RFIL, COREC, PREF, COGET, COKEY, OPERAT, Z-CLE.

sortie : RETCODE, RREF, RFIELD.

fonction :

- fournir dans RREF la référence de l'article contenu dans le fichier dont la référence est dans RFIL, et qui vérifie la condition "clé article OPERAT Z-CLE" sur la clé d'accès dont la référence est contenue dans COKEY; cet article sera du type dont la référence est contenue dans COREC, ou sera de type quelconque si COREC = 0.

- PREF doit contenir la référence du dernier article du fichier auquel on a accédé; si PREF = 0, on accède au premier qui vérifie la condition, sinon l'article auquel on accède suivra celui dont la référence est dans PREF.

- si COGET = 1, les valeurs d'itelem de l'article seront transférées dans RFIELD.

- diagnostics d'erreurs (transmis dans RETCODE) : les mêmes que ceux définis dans l'accès par clé aux articles d'un type.

* Accès séquentiel aux articles cibles d'un chemin.

entrée : COP = 35/36, SREF, COSET, COREC, OREF, COGET, PREF.

sortie : RREF, RFIL, RFIELD, RETCODE.

fonction :

- fournir dans RREF la référence de l'article (type=COREC) cible d'un chemin dont la référence de l'article origine est donnée dans OREF et le code du type (de chemin) dans COSET; sachant que la référence de l'article précédent est donnée dans PREF.

- fournir la référence du fichier contenant cet article et éventuellement ses valeurs d'item (suivant COGET).

- ordre : du premier au dernier si COP = 35
  du dernier au premier si COP = 36.

* Accès par clé aux articles-cibles d'un chemin.

entrée : COP = 37, SREF, COSET, COREC, OREF, COKEY, OPERAT, Z-CLE, COGET, PREF.

sortie : RREF, RFIL, RFIELD.

fonction :

- la même que pour l'accès séquentiel aux articles-cibles d'un chemin, avec en plus l'obligation pour la valeur de clé contenue dans l'article de vérifier la condition : "clé article OPERAT Z-CLE"

* Accès par référence à un article.

entrée : COP = 38, SREF, COREC, RREF.

sortie : RETCODE, RFIL, RFIELD.

fonction :

- si COGET = 1, fournir dans RFIELD les valeurs d'item de l'article dont la référence est contenue dans RREF, et dont le type est donné par COREC.

- garnir RFIL avec la référence du fichier qui contient cet article.

- cette primitive ne fonctionne que si RREF contient la référence du dernier article de ce type auquel on a accédé.
  RREF ne peut contenir la référence d'un article supprimé.

- diagnostics d'erreurs (transmis dans RETCODE) :

  "COREC invalide" si COREC ne contient pas le code d'un type d'article de la Base de Données.

  "RREF invalide" si RREF ne contient pas la référence du dernier article dont le type est contenu dans COREC auquel on ait accédé, ou si cet article a été supprimé.

Les primitives de modification.

1. Primitives ayant pour cible un fichier.

* Reinitialisation des fichiers.

  entrée : COP = 25, SREF, FILNAME.

  sortie : RFIL, RETCODE.

  fonction :

  - supprimer tous les enregistrements contenus dans le fichier dont le nom est contenu dans FILNAME;
    cette primitive ne fonctionne que sur des fichiers pour lesquels n'existe aucune clé d'accès;
    RFIL contiendra la référence du fichier.

  - le fichier dont le nom est contenu dans FILNAME doit être fermé.

  - diagnostics d'erreurs (transmis dans RETCODE) :

    "fichier et fonction incompatibles" si FILNAME contient le nom d'un fichier sur lequel au moins une clé d'accès est définie.

    "fichier en opération" si le fichier est déjà ouvert.

    "fichier non disponible" si FILNAME contient le nom d'un fichier non existant ou non accessible par le SGF.

2. Primitives ayant pour cible un article.

* Création d'un article.

  entrée : COP = 61, SREF, RFIL, COREC, Z-VALIT.

  sortie : RETCODE, RFIELD.

  fonction :

  - créer un article dans le fichier dont la référence est donnée dans RFIL;
    le type de cet article est contenu dans COREC et ses valeurs d'item dans Z-VALIT.

- cet article sera crée de façon à ce que les ordres d'accès définis sur le fichier soient respectés.
  Pour les fichiers sur lesquels sont définis des clés d'accès, l'insertion se fera en respectant la séquence des clés d'accès.
  Pour les autres fichiers, la création se fera en fin de fichier.

- diagnostics d'erreurs (transmis dans RETCODE) :

  "RFIL invalide" si RFIL ne contient pas la référence d'un fichier de la Base de Données.

  "COREC invalide" si COREC ne contient pas le code d'un type d'article de la Base de Données.

  "mode-ouverture invalide" si le fichier dont la référence est contenue dans RFIL est ouvert en consultation.

  "clé double" si la valeur de la clé identifiante de l'article à créer est déjà idemtifiante d'un article du fichier.

* Suppression d'un article.

entrée : COP = 62, SREF, COREC, RREF.

sortie : RETCODE.

fonction :

- supprimer du fichier auquel il appartient l'article dont le type est donné dans COREC et dont la référence est contenue dans RREF.

- la référence contenue dans RREF doit être celle du dernier article auquel on a accédé dans le fichier.

- la référence de l'article supprimé peut emcore être utilisée pour réaliser l'accès à l'article suivant, et pour cela uniquement.

- diagnostics d'erreurs (transmis dans RETCODE) :

  "COREC invalide" si COREC ne contient pas le code d'un type d'article de la Base de Données.

  "RREF invalide" si RREF ne contient pas la référence du dernier article du type donné auquel on a accédé.

* Modification des valeurs d'items d'un article.

entrée : COP = 71, SREF, COREC, RREF, Z-VALIT.

sortie : RETCODE.

fonction :

- remplacer les valeurs d'item de l'article dont la référence est dans RREF et le type dans COREC, par les valeurs d'item de l'article contenues dans Z-VALIT.

- pour les fichiers sur lesquel est définie une clé d'accès identifiante, cette clé ne fait pas partie des items modifiables; il faut procéder à la suppression et à la recréation de cet article.

- la référence contenue dans RREF doit être celle du dernier article auquel on a accédé dans le fichier.

- diagnostics d'erreurs (transmis dans RETCODE) :

  "COREC invalide" si COREC ne contient pas le code d'un type d'article de la Base de Données.

  "RREF invalide" si RREF ne contient pas la référence du dernier article dont le type est contenu dans COREC auquel on ait accédé, ou si cet article a été supprimé.

* Insertion d'un article dans un chemin.

entrée : COP = 81, SREF, COSET, OREF, RREF, SETLST, CURLST.

sortie : RETCODE

fonction :

- insérer l'article dont la référence est dans RREF, dans le chemin ayant pour origine l'article dont la référence est dans OREF et pour type celui dont le code est dans RREF;
  cette insertion doit se faire en fonction de l'ordre défini pour ce chemin, et/ou de l'article membre auquel on a déjà accédé dans ce chemin (donné par SETLST et CURLST).

* Retrait d'un article d'un chemin.

entrée : COP = 82, SREF, COSET, OREF, RREF.

sortie : RETCODE.

fonction :

- ôter d'un chemin (type = COSET, origine = OREF), l'article-cible dont la référence est contenue dans RREF.

Les primitives de contrôle.

Ces primitives ont pour objet de contrôler et commander les mécanismes chargés de la gestion de la concurrence et de la sécurité.

Ces fonctions nécessitant une étude dans un contexte plus large que celui du modèle Cobol, ne seront pas implémentées; des codes-opérations leur ont cependant été attribués : 51, 52, 53,...

Toute tentative d'accès à ces primitives se soldera par une valeur de RETCODE signifiant leur non-implémentation.

Des exemples de ces fonctions de contrôle seraient :

- demande de contrôle sur un article fonction :
  - vérifier l'existence d'un article;
  - fournir le type de l'article;
  - bloquer un article.
- demande de libération d'un article fonction ;
  - abandonner le blocage d'un article.

4. REALISATION DE L'INTERFACE DE LA BASE DE DONNEES BDTEST.

Dans les trois chapitres précédents nous avons définis plusieurs concepts:

1. le modèle d'accès établi pour décrire tout ensemble de données;
2. le modèle de données propre au Pascal MT+;
3. l'interface Base de Données établi dans le contexte IDML et qui a pour rôle de présenter à l'utilisateur un modèle de données et un langage de manipulation de ces données unique, quel que soit le Système de Gestion de Base de Données ou de Fichier qui gère l'ensemble des données.

Il faut maintenant établir des liens entre ces trois structures et voir comment elles peuvent agir ensemble vis-à-vis d'un sytème d'information.

Pour cela, nous allons montrer par un exemple concret comment un programme d'application met en pratique, à travers le modèle de données Pascal MT+ les éléments du modèle d'accès et comment il réagit aux primitives de l'interface IDML.

Nous allons tout d'abord décrire la structure des données (c'est-à-dire les objets de la Base de Données).

Puis nous spécifierons la réalisation de l'interface:

- les données internes qui servent à mémoriser des états ou des caractéristiques des objets de la Base de Données;
- les fonctions logiques (vérification de la cohérence d'une requête) et les fonctions physiques (ordres d'accès Pascal).
- les primitives;

La Base de Données que nous avons construite est un exemple d'école.
Le but de ce travail n'étant pas de générer une interface optimale pour le modèle de Données Pascal, les informations reprises ci-après ne constituent qu'un ensemble restreint de ce qui peut être réalisé. Il ne s'agira pas de tirer des conclusions générales quant à l'utilisation conjointe du modèle d'accès, du modèle Pascal et des interfaces IDML.

Nous verrons d'ailleurs dans les annexes concernant la Base de Données Terminal et la Base de Données des Fichiers Standards que d'autre options ont été prises afin de mieux répondre aux nécessités.

4.1. LA STRUCTURE DES DONNEES DE LA BASE DE DONNEES.

Pour chacun des objets de la Base de Données, nous allons procéder en trois temps. Tout d'abord, nous établirons la correspondance entre le modèle d'accès et la modèle des données Pascal MT+. Ensuite, nous étendrons par programme les possibilités du modèle Pascal, afin d'offrir à l'utilisateur une plus grande souplesse d'utilisation. Enfin, nous spécifierons les objets de la Base de Données qui nous sert d'application.

1. La Base de Données.

- modèle Pascal:

  La structure de données Pascal ne dépasse pas le niveau des fichiers; on ne peut donc pas dériver directement du modèle Pascal une structure équivalente à celle du modèle d'accès.

- par programme:

  Nous introduisons cependant une définition du concept "Base de Données" applicable dans le cadre du modèle Pascal: une Base de Données est constituée de tout ensemble de fichiers pour lesquels un administrateur de la Base de Données aura, pour des raisons d'organisation ou de fonctinnement, jugé opportun de disposer d'une interface unique qui prendra en charge tous les accès à ces fichiers.

  Un nom et un mot de passe seront attribués par l'administrateur de la Base de Données. Tout accès à la Base de Données ne sera autorisé que si le nom et le mot de passe sont corrects.

- exemple:

  le nom de la Base de Données : BDTEST
  le mot de passe : bonjour

2. Les fichiers.

- modèle Pascal:

  Suivant la norme Pascal, un fichier est toute collection d'enregistrements logiques de même taille susceptible d'être mise à la disposition f'un programme.

  A tout fichier est attribué un nom qui l'identifie parmi tous les fichiers connus du programme et qui le référencera lors de toute opération le concernant. La portée de ce nom est limitée au programme. L'identification d'un fichier parmi tous les fichiers du système sera réalisée par un mécanisme offert par le constructeur permettant de réaliser la correspondance entre le nom-programme et le nom-système d'un fichier.

L'utilisateur de l'interface connaît un fichier par son nom-programme.

Vis-à-vis de l'utilisateur, un seul mode d'accès est disponible : l'accès séquentiel. Les articles sont rangés à la suite l'un de l'autre dans l'ordre de leur création et ne sont accessibles que dans cet ordre.

- par programme:

Si l'accès offert à l'utilisateur est strictement séquentiel, la réalisation ne l'est pas pour autant.

Deux raisons ont amené à cette décision:

- les opérations disponibles sur un fichier séquentiel sont limitées: pas d'extension directe du fichier, pas de possibilite de modification ni de suppression sans utiliser de fichiers temporaires de travail, ce qui entraîne une très lourde charge pour le système.

- l'interface doit prendre en compte que la Base de Données est multi-utilisateurs. Chacun de ces utilisateurs peut accéder séquentiellement aux articles du fichier, sans se préoccuper des autres utilisateurs. Il faut dès lors que la fenêtre du fichier puisse se déplacer à volonté dans le fichier pour se positionner là où est arrivé chacun des utilisateurs.

C'est pourquoi, le programme considérera que les accès physiques à réaliser le seront de façon directe. La correspondance entre la référence d'article que donne l'utilisateur et la valeur de la clé d'accès direct est immédiate.

- exemple:

la Base de Données est constituée de deux fichiers :

- désignation : SIGN
  référence attribuée par le système : 1
  nom-système : b:SIGN.DAT
  type d'article qui le compose : PERSONNE

- désignation : DEP
  référence attribuée par le système : 2
  nom-système : b:DEP.DAT
  type d'article qui le compose : DEPTMT

3. Les types d'articles.

- modèle Pascal:

Le type d'article décrit les propriétés des articles qui appartiennent à ce type. Chaque type d'article possède un nom qui l'identifie parmi tous les types d'articles susceptibles d'être contenus dans la Base de Données. A un type d'article est toujours associé au moins un item.

- par programme:

Le modèle Pascal ne fournit aucune fonction permettant de supprimer un article d'un fichier. Cette suppression doit être organisée par le programme.

Il existe deux solutions pour détruire un article:

- destruction physique de l'article : réalisable par réorganisation du fichier (déplacement d'une position de tous les articles suivant l'article à supprimer) Cette solution n'est pas applicable ici car elle entraîne en cours de traitement une modification des références aux articles.

- destruction logique de l'article : un indicateur est positionné pour signaler si l'article est logiquement présent ou nom. Pour ce faire, un item a été ajouté à chaque type d'article. Il occupe un caractère et n'est pas connu ,donc accessible, par l'utilisateur.

- exemple:

Deux types d'articles ont été distingués:

- nom du type d'article : PERSONNE
  désignation : 1
  longueur : 63 caractères accessibles par l'utilisateur
  items constituants : NOM_E, NUM_E, SALAIRE, TRAVAIL

- nom du type d'article : DEPTMT
  désignation : 2
  longueur : 25 caractères accessibles par l'utilisateur
  items constituants : NOM_D, CODE_D, ADR, NB_EMPL

4. Les items.

- modèle Pascal:

Un item décrit les propriétés des valeurs d'item qui lui appartiennent; il est associé à un type d'article ou à un item décomposable. Les descriptions des items d'un type d'article se présentent dans l'ordre dynastique de la structure de décomposition. Un item porte un nom qui l'identifie parmi tous ceux ayant la même ascendance dans la structure de décomposition. Un item est toujours obligatoire; il peut être répétitif de répétitivité limitée.

- exemple:

Plusieurs items ont été dégagés. Aucune désignation ne leur a cependant été donnée car l'accès sur base de l'item n'a pas été réalisé.

- nom de l'item : NOM_E nom d'un employé
  propriétés : item élémentaire et simple
  valeur : caractères alpha-numériques
  longueur : 15 octets

- nom de l'item : NUM_E numéro d'un employé
  propriétés : item élémentaire et simple
  valeur : un entier positif [0..64534]
  longueur : 2 octets

- nom de l'item : SALAIRE salaire d'un employé
  propriétés : item élémentaire et simple
  valeur : un entier positif [0..64534]
  longueur : 2 octets

- nom de l'item : TRAVAIL
  propriétés : item décomposable et de répétitivité 2
  composants : DPTMT, PROJET
  longueur : 12 caractères

- nom de l'item : DPTMT code du département
  propriétés : item élémentaire et simple
  valeur : caractères alpha-numériques
  longueur : 2 octets

- nom de l'item : PROJET libellé d'un projet
  propriétés : item élémentaire et de répétitivité 2
  valeur : caractères alpha-numériques
  longueur : 10 octets

- nom de l'item : NOM_D libellé du département
  propriétés : item élémentaire et simple
  valeur : caractères alpha-numériques
  longueur : 10 octets

- nom de l'item : CODE_D code du département
  propriétés : item élémentaire et simple
  valeur : caractères alpha-numériques
  longueur : 2 octets

- nom de l'item : ADR localisation du département
  propriétés : item élémentaire et simple
  valeur : caractères alpha-numériques
  longueur : 10 octets

- nom de l'item : NB_EMPL nombre d'employés occupés dans le cadre d'un projet du département
  propriétés : item élémentaire et de répétitivité 2
  valeur : un entier positif [0..64534]
  longueur : 2 octets

4.2. LES VARIABLES DE GESTION INTERNE.

Ces données internes servent à mémoriser des états ou des caractéristiques des objets de la Base de Données de manière globale ou vis-à-vis de chaque utilisateur en particulier. C'est sur ces données que l'interface se base pour réaliser l'exécution logique des primitives.

1. La Base de Données.

   NOUV_BD : indique, à tout moment, le nombre d'ouvertures de la Base de Données qui sont en cours.

   NFICH : contient le nombre de fichiers qui constituent la Base de Données.

2. Les fichiers.

   Les données qui suivent concernent les fichiers; chacune d'entre elles est présente pour chacun des fichiers et doit donc être indicée par un identificateur de fichier (qui est égal à la référence du fichier).

   NOUV_FI : mémorise le nombre d'ouvertures demandées sur le fichier.

   PROT : renseigne le mode d'ouverture logique du fichier : consultation, mise-à-jour, consultation exclusive, mise-à-jour exclusive.

   FIN_FICH : indique la taille maximum que peut avoir le fichier (exprimé en nombre d'enregistrements)

   LAST_ID : contient la dernière référence attribuée à un enregistrement du fichier.

   CURRENT : renseigne la référence de l'enregistrement actuellement dans la zone tampon du fichier.

   MODIF : signale si l'enregistrement qui se trouve dans la zone tampon du fichier est une copie modifiée ou non de l'enregistrement qui lui correspond dans le fichier.

3. La situation des utilisateurs.

   Les informations qui suivent concernent le travail de chaque utilisateur; chacune d'entre elles est présente pour chacun des utilisateurs et doit donc être indicée par un identificateur d'utilisateur (la référence interne de l'utilisateur qui est attribuée par le système IDML)

   MO_FICH : indique si le mode d'ouverture des fichiers est normal (ouverture par l'utilisateur obligatoire) ou s'il est automatique

(l'utilisateur ignore la notion de fichier).

NBOUV_FI : contient, pour chaque fichier, le nombre d'ouvertures opérées par l'utilisateur.

LRUN : renseigne, pour chaque fichier, la référence du dernier article auquel l'utilisateur a accédé (référence courante pour l'utilisateur).

4.3. LES FONCTIONS INTERNES.

Nous avons adopté la convention suivante en ce qui concerne le vocabulaire : un paramètre a une valeur correcte si il respecte les règles dans un domaine déterminé; un paramètre a une valeur valide si il présente les conditions requises pour produire son effet.
Un paramètre valide est donc nécessairement correct, mais un paramètre correct peut être non valide pour une opération donnée.

*** INIT_TEST_BD ***

- fonction : initialiser les différentes données internes à l'interface

*** BDTEST_ERREUR ***

- entrée : CODE_ERR
- sortie : RETCODE
- fonction : assigner dans RETCODE la valeur de CODE_ERR

*** BDVERIF ***

- sortie : valeur booléenne
- fonction : vérifier si la Base de Données est ouverte
- messages d'erreur :

  7. La Base de Données n'est pas ouverte

*** FIVERIF ***

- entrée : FILNAME
- sortie : RFIL, valeur booléenne
- fonction : vérifier si le nom contenu dans FILNAME correspond à un fichier de la Base de données.
- messages d'erreur :

  8. Le fichier est inconnu dans cette Base de Données

*** RFILVERIF ***

- entrée : RFIL
- sortie : RETCODE, valeur true ou false
- fonction : vérifier si RFIL contient une valeur correcte de référence à un fichier de la BD
- messages d'erreur :

  9. RFIL n'est pas la référence à un fichier de cette BD

*** ARTVERIF ***

- entrée : COREC
- sortie : valeur booléenne
- fonction : vérifier si COREC est le code d'un type d'article de la BD
- messages d'erreur :

  10. COREC n'est pas le code d'un type d'article de cette BD

*** GETVERIF ***

- entrée : COGET
- sortie : valeur booléenne
- fonction : vérifier si COGET contient un code correct d'accès aux items
- messages d'erreur :

  11. COGET doit être égal à 0 (aucun ou 1 (tous les items

*** F_GET_REC ***

- entrée : REFILE = référence correcte d'un fichier,
  IDELEM = référence d'un article physiquement dans le fichier
- sortie : CODE_ERR, zone tampon
- fonction : transférer dans la zone tampon du fichier REFILE l'article de clé système IDELEM

*** F_OPEN ***

- entrée : REFILE = référence correcte d'un fichier de la BD
- fonction : ouvrir physiquement le fichier de référence REFILE en lecture/ modification

*** F_CLOSE ***

- entrée : REFILE = référence correcte d'un fichier physiquement ouvert de la BD,

- fonction : fermer physiquement un fichier ouvert de la BD

*** FERM_FICH ***

- entrée : REFILE = référence correcte d'un fichier ouvert,
  CAS = portée de cette opération (totale ou partielle)

- fonction : réaliser la fermeture logique, totale ou partielle, pour un utilisateur, d'un fichier ouvert;
  si nécessaire demander une fermeture physique.

*** OUVRIR_FICH ***

- entrée : REFILE = référence correcte d'un fichier de la BD,
  MODE = indication valide du type d'ouverture à réaliser

- fonction : effectuer une ouverture logique et physique si nécessaire du fichier ,et ce dans le mode indiqué.

*** OUVVERIF ***

- entrée : REFILE = référence correcte d'un fichier de la BD,
  OPERATION = lecture ou écriture

- sortie : valeur booléenne

- fonction : vérifier si l'utilisateur peut accéder au fichier pour effectuer l'opération donnée.

- messages d'erreur :

  13. Le fichier n'est pas ouvert 14. Le fichier est en opération dans un mode incompatible

*** DMDE_OUVERTURE ***

- entrée : REFILE = référence correcte d'un fichier de la BD,
  MODE = protection désirée sur le fichier

- sortie : RETCODE

- fonction : effectuer, si possible, l'ouverture du fichier REFILE dans le MODE voulu pour l'utilisateur qui en a fait la demande.

- messages d'erreur :

  14. Le fichier est en opération dans un mode incompatible

*** ART_ARG ***

- entrée : REFILE = référence correcte d'un fichier de la BD, COGET valide, la zone tampon
- sortie : RREF, RFIELD
- fonction : tranférer le contenu de la zone tampon du fichier REFILE ("article") dans la zone RFIELD ("argument")

*** ARG_ART ***

- entrée : REFILE = référence correcte d'un fichier de la BD, ID = référence à attribuer à l'article à transférer, RFIELD
- sortie : RREF, zone tampon
- fonction : transférer le contenu de RFIELD ("argument") dans la zone tampon du fichier REFILE ("article")

*** ART_IN_FILE ***

- entrée : REFILE = référence correcte d'un fichier de la BD, IDELEM = référence correcte d'un article du fichier
- sortie : valeur booléenne
- fonction : vérifier que la référence IDELEM est valide

*** ART_SUPP ***

- entrée : REFILE = référence correcte d'un fichier de la BD, zone tampon contenant un article
- sortie : valeur booléenne
- fonction : vérifier si l'article physiquement présent dans la zone tampon est logiquement supprimé

*** RECH_ELEM ***

- entrée : REFILE = référence correcte d'un fichier accesible en lecture de la BD,
  IDELEM = la référence potentielle de l'article à trouver
- sortie : RETCODE, zone tampon, valeur booléenne
- fonction : accéder si possible à l'article de référence IDELEM dans le fichier REFILE
- messages d'erreur :

  12. L'article demandé n'existe pas
  22. L'article demandé n'existe plus

*** RECH_PLACE ***

- entrée : REFILE = référence correcte d'un fichier de la BD
- sortie : NEWID, valeur booléenne
- fonction : vérifier s'il y a de la place dans le fichier REFILE pour créer un nouvel article et fournir la référence attribuée à ce nouvel article.

4.4. SPECIFICATIONS DES PRIMITIVES.

Nous décrivons ci-dessous les primitives qui constituent l'interface.

Elles font appel aux fonctions logiques et physiques énoncées ci-avant. Dans la description qui va suivre, les noms de ces fonctions seront écrits en majuscule entre les symboles [et].
De même, toute opération élémentaire ou test sera inscrit entre [].

Pour chaque primitive, nous citons les variables en entrée, les variables en sortie et la réalisation.

*** OUVERTURE DE LA BASE DE DONNEES ***

- entrée : COP = 11, SSNAME, PSW, PROTECT

- sortie : RETCODE

- réalisation :

  - si SSNAME <> "BDTEST" alors [ERREUR 4] , ARRET
  - si PSW <> "bonjour" alors [ERREUR 5] , ARRET
  - si PROTECT <> 1 ou 2 alors [ERREUR 6] , ARRET

  - [ajouter 1 à NOUV_BD]

  - si première ouverture pour l'utilisateur
    [MO_FICH (user) = 0] ?

    alors mise-à-jour du mode d'ouverture des fichiers
    [MO_FICH (user) <-- PROTECT]

    sinon rappeler à l'utilisateur le mode d'ouverture
    des fichiers établi lors de la 1ere ouverture
    [PROTECT <-- MO_FICH (user)]

- messages d'erreurs :

  4. La Base de Données est inconnue

  5. Le mot de passe est incorrect

  6. Le mode d'ouverture des fichiers PROTECT est incorrect

*** FERMETURE DE LA BASE DE DONNEES ***

- entrée : COP = 12

- sortie : RETCODE

- réalisation :

  - si la Base de Données n'est pas ouverte alors ARRET
    [BDVERIF false] ?

  - [soustraire 1 de NOUV_BD]

  - fermer les fichiers que l'utilisateur a omis de fermer
    [FERM_FICH (totalement)]

*** OUVERTURE DES FICHIERS ***

- entrée : COP = 21 ou 22, FILNAME, PROTECT

- sortie : RETCODE, RFIL

- réalisation :

  - si la Base de Données n'est pas ouverte alors ARRET
    [BDVERIF false] ?

  - COP = 21 : . ouvrir tous les fichiers
    [DMDE_OUVERTURE (PROTECT)]

    . si un fichier ne peut pas être ouvert
    [RETCODE <> 0] ?

    alors fermer tous les fichiers que l'on vient d'ouvrir
    [FERM_FICH (partiellement)]

  - COP = 22 : . si FILNAME correspond à un fichier de la BD
    [FIVERIF true] ?

    alors ouvrir le fichier
    [DMDE_OUVERTURE (PROTECT)]

*** FERMETURE DES FICHIERS ***

- entrée : COP = 23 ou 24, RFIL

- sortie : RETCODE

- réalisation :

  - si la Base de Données n'est pas ouverte alors ARRET
    [BDVERIF false] ?

  - COP = 23 : . tous les fichiers ouverts par l'utilisateurs doivent être fermés
    [FERM_FICH (totalement)]

  - COP = 24 : . si RFIL est correct
    [RFILVERIF true] ?

    et si l'utilisateur a ouvert le fichier
    [NBOUV_FI (user,RFIL) > 0] ?

    alors fermer le fichier
    [FERM_FICH (partiellement)]

*** ACCES SEQUENTIEL AUX ARTICLES D'UN TYPE DONNE ***

- entrée : COP = 31, COREC, COGET, PREF

- sortie : RETCODE, RFIL, RREF, RFIELD

- réalisation :

  - si la Base de Données n'est pas ouverte alors ARRET
    [BDVERIF false] ?
  - si COREC n'est pas correct alors ARRET
    [ARTVERIF false] ?
  - si COGET n'est pas correct alors ARRET
    [GETVERIF false] ?

  - mettre dans RFIL la référence du fichier qui comprend les articles de type COREC
    [RFIL <-- COREC]

  - si l'utilisateur peut accéder au fichier en lecture
    [OUVVERIF (lect) true] ?

    alors . chercher l'élément qui suit l'élément de référence PREF
    [elem <-- PREF + 1]

    . tant que

    l'accès fournit un article supprimé
    [RECH_ELEM false et RETCODE = 22] ?
    et que la fin du fichier n'est pas atteinte
    [RECH_ELEM false et RETCODE <> 22] ?

    demander l'accès à l'article suivant
    [ajouter 1 à elem]

    . si un article a été lu
    [RETCODE = 0] ?

    alors transférer la zone tampon dans RFIELD
    [ART_ARG]

*** ACCES SEQUENTIEL AUX ARTICLES D'UN FICHIER ***

- entrée : COP = 33, RFIL, COGET, PREF

- sortie : RETCODE, COREC, RREF, RFIELD

- réalisation :

  - si la Base de Données n'est pas ouverte alors ARRET
    [BDVERIF false] ?
  - si RFIL n'est pas correct alors ARRET
    [RFILVERIF false] ?
  - si COGET n'est pas correct alors ARRET
    [GETVERIF false] ?

  - mettre dans COREC le code du type d'article qui est contenu dans le fichier de référence RFIL
    [COREC <— RFIL]

  - si l'utilisateur peut accéder au fichier en lecture
    [OUVVERIF (lect) true] ?

    alors . chercher l'élément qui suit l'élément de référence PREF
    [elem <— PREF + 1]

    . tant que

    l'accès fournit un article supprimé
    [RECH_ELEM false et RETCODE = 22] ?
    et que la fin du fichier n'est pas atteinte
    [RECH_ELEM false et RETCODE <> 22] ?

    demander l'accès à l'article suivant
    [ajouter 1 à elem]

    . si un article a été lu
    [RETCODE = 0] ?

    alors transférer la zone tampon dans RFIELD
    [ART_ARG]

*** ACCES PAR REFERENCE A UN ARTICLE ***

- entrée : COP = 38, COREC, RREF

- sortie : RETCODE, RFIL, RFIELD

- réalisation :

  - si la Base de Données n'est pas ouverte alors ARRET
    [BDVERIF false] ?
  - si COREC n'est pas correct alors ARRET
    [ARTVERIF false] ?

  - mettre dans COREC le code du type d'article qui est contenu dans le fichier de référence RFIL
    [COREC <-- RFIL]

  - signaler qu'on veut recevoir les valeurs de tous les items
    [COGET <-- 1]

  - si l'utilisateur peut accéder au fichier en lecture
    [OUVVERIF (lect) true] ?

    alors si RREF est la référence courante, pour l'utilisateur, d'un article de type COREC
    [RREF = LRUN (COREC)] ?

      alors si l'élément existe toujours
      [RECH_ELEM true] ?
      alors transférer la zone tampon dans RFIELD
      [ART-ARG]

      sinon [ERREUR 15]

- messages d'erreurs:

  15. RREF n'est pas la référence courante de l'article

*** REINITIALISATION D'UN FICHIER ***

- entrée : COP = 25, FILNAME

- sortie : RETCODE, RFIL

- réalisation :

  - si la Base de Données n'est pas ouverte alors ARRET
    [BDVERIF false] ?
  - si FILNAME ne correspond pas à un fichier de la BD alors ARRET
    [FIVERIF false] ?

  - si le fichier est physiquement fermé
    [NOUV_FI (RFIL) = 0] ?

    alors . générer physiquement un nouveau fichier,
    [ASSIGN], [REWRITE]

    . créer le premier article qui contient la valeur de la dernière référence attribuée à un article du fichier
    [LAST_ID (RFIL) <-- 0], [SEEKWRITE 0]

    . créer le dernier article du fichier afin de réserver sur disque toute la place nécessaire
    [SEEKREAD fin-fich], [CLOSE]

    sinon [ERREUR 16]

- messages d'erreurs:

  16. Le fichier à réinitialiser est actuellement en opération

*** CREATION D'UN ARTICLE ***

- entrée : COP = 61, RFIL, COREC, Z_VALIT

- sortie : RETCODE, RFIELD

- réalisation :

  - si la Base de Données n'est pas ouverte alors ARRET
    [BDVERIF false] ?
  - si RFIL n'est pas correcte alors ARRET
    [RFILVERIF false] ?
  - si COREC n'est pas correct alors ARRET
    [ARTVERIF false] ?

  - si le fichier de référence RFIL ne comprend pas
    des articles de type COREC
    [RFIL <> COREC] ?

    alors [ERREUR 17]

    sinon si l'utilisateur peut accéder au fichier en écriture
    [OUVVERIF (ecr) true] ?

    alors si il y a encore de la place dans le fichier
    [RECH_PLACE]

    alors transférer Z_VALIT dans la zone tampon
    [ARG_ART]

    sinon [ERREUR 18]

- messages d'erreurs:

  17. Le code de type d'article et la référence du fichier ne sont pas compatibles

  18. Il n'y a plus de place dans le fichier pour créer un nouvel article

*** MODIFICATION DES VALEURS D'ITEMS D'UN ARTICLE ***

- entrée : COP = 71, COREC, RREF, Z_VALIT

- sortie : RETCODE, RFIL, RFIELD

- réalisation :

  - si la Base de Données n'est pas ouverte alors ARRET
    [BDVERIF false] ?
  - si COREC n'est pas correct alors ARRET
    [ARTVERIF false] ?

  - mettre dans RFIL la référence du fichier qui comprend les articles de type COREC
    [RFIL <-- COREC]

  - si l'utilisateur peut accéder au fichier en écriture
    [OUVVERIF (ecr) true] ?

    alors si RREF est la référence courante, pour l'utilisateur, d'un article de type COREC
    [RREF = LRUN (COREC)] ?

      alors si l'article est toujours présent dans le fichier
      [RECH_ELEM true] ?
      alors transférer le contenu de Z_VALIT dans la zone tampon
      [ART-ARG]

      sinon [ERREUR 15]

- messages d'erreurs:

  15. RREF n'est pas la référence courante de l'article

*** SUPPRESSION D'UN ARTICLE ***

- entrée : COP = 62, COREC, RREF

- sortie : RETCODE, RFIL

- réalisation :

  - si la Base de Données n'est pas ouverte alors ARRET
    [BDVERIF false] ?
  - si COREC n'est pas correct alors ARRET
    [ARTVERIF false] ?

  - mettre dans RFIL la référence du fichier qui comprend les articles de type COREC
    [RFIL <-- COREC]

  - si l'utilisateur peut accéder au fichier en écriture
    [OUVVERIF (ecr) true] ?

    alors si RREF est la référence courante, pour l'utilisateur, d'un article de type COREC
    [RREF = LRUN (COREC)] ?

      alors . accéder physiquement à l'article
      [F_GET_REC (RREF)]

      . mettre à jour dans l'article l'indicateur de suppression
      [1ere position de l'article <-- 1]

      . signaler que l'article dans la zone tampon a été modifié
      [MODIF (COREC) <-- vrai]

      sinon [ERREUR 15]

- messages d'erreurs:

  15. RREF n'est pas la référence courante de l'article

5. REALISATION DE L'INTERFACE DE LA BASE TERMINAL

5.1. LA STRUCTURE DES DONNEES DE LA BASE DE DONNEES.

1. La Base de Données.

- modèle Pascal:

La structure de données Pascal ne dépasse pas le niveau des fichiers; on ne peut donc pas dériver directement du modèle Pascal une structure équivalente à celle du modèle d'accès.

- par programme:

Nous introduisons cependant une définition du concept "Base de Données" applicable dans le cadre du modèle Pascal: une Base de Données est constituée de tout ensemble de fichiers pour lesquels un administrateur de la Base de Données aura, pour des raisons d'organisation ou de fonctinnement, jugé opportun de disposer d'une interface unique qui prendra en charge tous les accès à ces fichiers.

Un nom a été attribué à la Base de Données. Tout accès à la Base de Données ne sera autorisé que si le nom est correct.

- exemple:

le nom de la Base de Données : BDIO

2. Les fichiers.

- modèle Pascal:

A tout fichier est attribué un nom qui l'identifie parmi tous les fichiers connus du programme et qui le référencera lors de toute opération le concernant. La portée de ce nom est limitée au programme. L'identification d'un fichier parmi tous les fichiers du système sera réalisée par un mécanisme offert par le constructeur permettant de réaliser la correspondance entre le nom-programme et le nom-système d'un fichier.

L'utilisateur de l'interface connaît un fichier par son nom-programme.

Les fichiers qui vont être considérés dans cette Base de Données sont essentiellement des fichiers de texte. C'est pourquoi ils sont uniquement accessibles en séquentiel.

- par programme :

Vis-à-vis de l'utilisateur, le clavier et l'écran sont considérés, au même titre que les fichiers batch comme des fichiers de texte. Les primitives d'accès sont par ailleurs identiques.

- exemple:

la Base de Données est constituée de quatre fichiers :

- désignation : TIN
  référence attribuée par le système : 1
  nom-système : "clavier"
  type d'article qui le compose : texte
  opération disponible : lecture

- désignation : TOUT
  référence attribuée par le système : 2
  nom-système : "ecran"
  type d'article qui le compose : texte
  opération disponible : lecture

- désignation : BIN
  référence attribuée par le système : 3
  nom-système : FIDML.IN
  type d'article qui le compose : texte
  opération disponible : écriture

- désignation : BOUT
  référence attribuée par le système : 4
  nom-système : FIDML.OUT
  type d'article qui le compose : texte
  opération disponible : écriture

3. Les types d'articles.

- modèle Pascal:

Le type d'article décrit les propriétés des articles qui appartiennent à ce type. Les articles qui appartiennent au type texte peuvent être des caractères, des entiers, des réels ou des booléens.

- par programme:

Le programme considère l'article comme une chaîne de caractères terminées par le caractère Ascii ‹NULL›, sans distinction du contenu de ces caractères. Seul un symbole particulier peut être distingué. Il s'agit du symbole de fin de ligne.

- exemple :

Le seul type d'article reconnu est donc la chaîne de caractères.
La longueur de cette chaîne est spécifiée par le code de type d'article. Les codes corrects appartiennent à l'intervalle [0..132].

Le code particulier 0 signifie que la longueur de la chaîne est déterminée par la présence du symbole de fin de ligne, avec un maximum de 132 caractères.

5.2. LES VARIABLES DE GESTION INTERNE.

Ces données internes servent à mémoriser des états ou des caractéristiques des objets de la Base de Données. C'est sur ces données que l'interface se base pour réaliser l'exécution logique des primitives.

Contrairement à l'interface analysée au chapitre précédent (BDTEST), cette interface ne supporte la présence que d'un utilisateur à la fois.

1. La Base de Données.

   IO_NOUV_BD : indique, à tout moment, le nombre d'ouvertures de la Base de Données qui sont en cours.

   IO_NFICH : contient le nombre de fichiers qui constituent la Base de Données.

2. Les fichiers.

   Les informations liées à un fichier sont en nombre réduit du fait que l'organisation et l'accès aux articles du fichier sont purement séquentielle, tant du point de vue de l'utilisateur que du programme.

   Dès lors, la notion de référence d'article perd ici toute sa valeur.

   Quelle que soit la référence donnée par l'utilisateur, l'accès à l'article suivant est conditionné par l'emploi des primitives de travail interactif avec le clavier et l'écran ou des primitives d'accès séquentiel dans un fichier texte.

   La seule information nécessaire concerne le nombre d'ouvertures des fichiers

   IO_ETAT (fichier)

5.3. LES FONCTIONS INTERNES.

*** INIT_IO_BD ***

- fonction : initialiser les différentes données internes à l'interface

*** BDIO_ERREUR ***

- entrée : CODE_ERR
- sortie : RETCODE
- fonction : assigner dans RETCODE la valeur de CODE_ERR

*** TERMINAL ***

- sortie : valeur booléenne
- fonction : signaler si le terminal (clavier/écran) de l'utilisateur est connecté.

Cette fonction permet de savoir s'il faut travailler de manière interactive directe ou non (dans ce cas il s'agit de considérer les fichiers batch comme outil de communication).

*** F_OPEN ***

- entrée : REFILE = référence valide d'un fichier de la BD
- fonction : ouvrir logiquement et physiquement si nécessaire le fichier de référence REFILE.

Le mode d'ouverture physique est indiqué par la référence du fichier:

- REFILE = 1 : ouverture du clavier (en lecture); apparition du symbole ">> " sur l'écran signalant que le système est prêt à recevoir des articles venant du clavier. Dans ce cas, l'écran n'est pas considéré comme le fichier d'écriture mais uniquement comme support de l'écho du clavier.
- REFILE = 2 : ouverture de l'écran (en écriture) qui permet au système de communiquer des articles à l'utilisateur.
- REFILE = 3 : ouverture du fichier batch en lecture; [RESET] du fichier 'FIDML.IN'.
- REFILE = 4 : ouverture du fichier batch en écriture; [REWRITE] du fichier 'FIDML.OUT'

*** F_OUVERTURE ***

- entrée : REFILE : référence correcte d'un fichier de la BD
- sortie : RETCODE, RFIL
- fonction : examiner la validité de la référence REFILE du fichier et de la protection demandée PROTECT;
  ouvrir si possible ce fichier.

Si le fichier renseigné est le clavier ou l'écran, il faut vérifié si celui-çi est connecté au système. Si ce n'est pas le cas, cette procédure va aiguiller l'utilisateur de l'interface vers le fichier batch correspondant.

Quant aux modes d'ouvertures, ils seront valides s'ils sont compatibles avec le fichier (lecture du clavier ou du fichier BIN, écriture de l'écran ou du fichier BOUT).

*** F_CLOSE ***

- entrée : REFILE = référence valide d'un fichier da la BD
- fonction : fermer physiquement le fichier de référence REFILE.

Il s'agit soit du fichier batch en lecture ou du fichier batch en écriture.

*** F_FERMETURE ***

- entrée : REFILE = référence correcte d'un fichier de la BD
- fonction : si le fichier est ouvert, le fermer logiquement de façon totale ou partielle;
  fermeture physique si nécessaire.

5.4. SPECIFICATIONS DES PRIMITIVES.

*** OUVERTURE DE LA BASE DE DONNEES ***

- entrée : COP = 11, SSNAME
- sortie : RETCODE
- réalisation :
  - si SSNAME <> "BDIO" alors [ERREUR 4]
  - [ajouter 1 à NOUV_BD]
- messages d'erreurs :

  4. La Base de Données est inconnue

*** FERMETURE DE LA BASE DE DONNEES ***

- entrée : COP = 12
- sortie : RETCODE
- réalisation :
  - si la Base de Données n'est pas ouverte alors [ERREUR 7]
    [IO_NOUVBD = 0] ?
  - [soustraire 1 de IO_NOUVBD]
  - si la Base de Données est totalment fermée
    [IO_NOUVBD = 0] ?
    alors fermer les fichiers que l'utilisateur a omis de fermer
    [F_FERMETURE (totalement)]
- messages d'erreur :

  7. La Base de Données n'est pas ouverte

*** OUVERTURE D'UN FICHIER ***

- entrée : COP = 22, FILNAME, PROTECT
- sortie : RETCODE, RFIL
- réalisation :
  - si la Base de Données n'est pas ouverte alors [ERREUR 7]
    [IO_NOUVBD = 0] ?
  - si FILNAME comprend le nom d'un fichier de la BD,
    [FILNAME = "TIN" ou "TOUT" ou "BIN" ou "BOUT "] ?
    alors demander l'ouverture
    [F_OUVERTURE]
    sinon [ERREUR 8]

- messages d'erreur :

  7. La Base de Données n'est pas ouverte
  8. Le fichier est inconnu dans cette Base de Données

*** FERMETURE DE TOUS LES FICHIERS ***

- entrée : COP = 23, RFIL

- sortie : RETCODE

- réalisation :

  - si la Base de Données n'est pas ouverte alors [ERREUR 7]
    [IO_NOUVBD = 0] ?

  - tous les fichiers ouverts par l'utilisateur doivent être fermés partiellement
    [F_FERMETURE]

- messages d'erreur :

  7. La Base de Données n'est pas ouverte

*** ACCES SEQUENTIEL AUX ARTICLES D'UN FICHIER ***

- entrée : COP = 33, RFIL, COREC

- sortie : RETCODE, RFIELD

- réalisation :

  - si la Base de Données n'est pas ouverte alors [ERREUR 7]
    [IO_NOUVBD = 0] ?
  - si RFIL n'est pas valide alors [ERREUR 20]
    [RFIL <> 1 et 3] ?
  - si le fichier n'est pas ouvert alors [ERREUR 13]
    [IO_ETAT (RFIL) = 0] ?

  - accéder à l'article suivant du fichier

  - si la fin du fichier est atteinte
    alors [ERREUR 12]
    sinon transférer l'article dans RFIELD
    (longueur définie par COREC)

- messages d'erreur :

  7. La Base de Données n'est pas ouverte
  12. L'article demandé n'existe pas
  13. Le fichier n'est pas ouvert
  20. Le fichier doit être ouvert en lecture

*** CREATION D'UN ARTICLE ***

- entrée : COP = 61, RFIL, COREC, Z_VALIT

- sortie : RETCODE, RFIELD

- réalisation :

  - si la Base de Données n'est pas ouverte alors [ERREUR 7]
    [IO_NOUVBD = 0] ?
  - si RFIL n'est pas valide alors [ERREUR 21]
    [RFIL <> 2 et 4] ?
  - si le fichier n'est pas ouvert alors [ERREUR 13]
    [IO_ETAT (RFIL) = 0] ?

  - transférer dans le fichier l'article contenu dans Z_VALIT
    (la longueur de l'article est déterminée par COREC)

- messages d'erreur :

  7. La Base de Données n'est pas ouverte
  13. Le fichier n'est pas ouvert
  21. Le fichier doit être ouvert en écriture

6. REALISATION DE L'INTERFACE DE LA BASE DES FICHIERS STANDARD.

6.1. LA STRUCTURE DES DONNEES DE LA BASE DE DONNEES.

1. La Base de Donnèes.

- modèle Pascal:

La structure de donnèes Pascal ne dèpasse pas le niveau des fichiers; on ne peut donc pas dèriver directement du modèle Pascal une structure èquivalente à celle du modèle d'accès.

- par programme:

Nous introduisons cependant une dèfinition du concept "Base de Donnèes" applicable dans le cadre du modèle Pascal: une Base de Donnèes est constituèe de tout ensemble de fichiers pour lesquels un administrateur de la Base de Donnèes aura, pour des raisons d'organisation ou de fonctinnement, jugè opportun de disposer d'une interface unique qui prendra en charge tous les accès à ces fichiers.

Un nom a ètè attribuè à la Base de Donnèes. Tout accès à la Base de Donnèes ne sera autorisè que si le nom est correct.

- exemple:

le nom de la Base de Donnèes : BDFI

2. Les fichiers.

- modèle Pascal:

A tout fichier est attribuè un nom qui l'identifie parmi tous les fichiers connus du programme et qui le rèfèrencera lors de toute opèration le concernant. La portèe de ce nom est limitèe au programme. L'identification d'un fichier parmi tous les fichiers du système sera rèalisèe par un mècanisme offert par le constructeur permettant de rèaliser la correspondance entre le nom-programme et libe nom-système d'un fichier.

La Base de Donnèes qui est sous le contrôle de cette interface est dynamique : les fichiers qui la constituent sont dèsignès par l'utilisateur pendant son travail.

L'utilisateur de l'interface dèsigne un fichier par son nom-système.

Les fichiers utilisès sont des fichiers dont le type des èlèments n'est pas prècisè. Cette option a ètè prise afin de laisser entière libertè à l'utilisateur quant au contenu et à l'organisation du fichier.

Néanmoins, tous les accès seront de type séquentiel.

- par programme :

Le fichier est constitué d'un ensemble d'octets. Le contenu de ces octets est laissé à la discrétion de l'utilisateur. La possibilité lui est donnée de subdiviser le fichier en lignes sans pour autant que le fichier soit de type texte (aucune conversion Ascii/Binaire n'est effectuée).

- exemple:

la Base de Données est constituée de quatre fichiers :

- désignation : FIN1
  référence attribuée par le système : 1
  nom-système : ?
  type d'article qui le compose : suite d'octets
  opération disponible : lecture

- désignation : FIN2
  référence attribuée par le système : 2
  nom-système : ?
  type d'article qui le compose : suite d'octets
  opération disponible : lecture

- désignation : FOUT1
  référence attribuée par le système : 3
  nom-système : ?
  type d'article qui le compose : suite d'octets
  opération disponible : écriture

- désignation : FOUT2
  référence attribuée par le système : 4
  nom-système : ?
  type d'article qui le compose : suite d'octets
  opération disponible : écriture

3. Les types d'articles.

- modèle Pascal:

Le type d'article décrit les propriétés des articles qui appartiennent à ce type. Ici, le type d'article n'est pas précisé. D'une manière générale, il s'agit de suite d'octets.

- par programme:

Le programme considère l'article comme une suite de caractetères terminée par le caractère Ascii <NULL>, sans distinction du contenu de ces caractères. Seul un symbole particulier peut être distingué. Il s'agit du symbole de fin de ligne.

- exemple :

Le seul type d'article reconnu est donc la chaîne de caractères.
La longueur de cette chaîne est spécifiée par le code de type d'article. Les codes corrects appartiennent à l'intervalle [0..255].

Le code particulier 0 signifie que la longueur de la chaîne est déterminée par la présence du symbole de fin de ligne, avec un maximum de 255 caractères.

6.2. LES VARIABLES DE GESTION INTERNE.

Ces données internes servent à mémoriser des états ou des caractéristiques des objets de la Base de Données. C'est sur ces données que l'interface se base pour réaliser l'exécution logique des primitives.

A nouveau, cette interface ne peut contrôler que le travail d'un seul utilisateur à la fois.

1. La Base de Données.

   FI_NOUV_BD : indique, à tout moment, le nombre d'ouvertures de la Base de Données qui sont en cours.

   FI_NFICH : contient le nombre de fichiers qui constituent la Base de Données.

2. Les fichiers.

   Si la gestion apparaît comme étant purement séquentielle pour l'utilisateur, il n'en reste pas moins que le type de gestion choisi (fichiers sans type) oblige l'interface à gérer elle-même la zone tampon de chacun des fichiers (l'accès au fichiers sans type se fait sur base de tableaux d'octets d'une taille multiple entier de 128)

   pour chaque fichier de la Base de Données

   FI_ETAT : indique si le fichier est ouvert ou non. Les ouvertures multiples ne sont pas prises en considération (l'interface ne réalise aucune analyse sur les noms-systèmes des fichiers que l'utilisateur lui demande d'ouvrir)

   BUFFER : est utilisé comme zone tampon de 256 positions

   CURRENT : pointe vers l'octet du buffer où devra se faire le prochain accès (prochain octet à lire ou prochaine position à écrire)

6.3. LES FONCTIONS INTERNES.

*** INIT_FI_BD ***

- fonction : initialiser les différentes données internes à l'interface

*** BDFI_ERREUR ***

- entrée : CODE_ERR
- sortie : RETCODE
- fonction : assigner dans RETCODE la valeur de CODE_ERR

*** F_OUVERTURE ***

- entrée : OPERATION = indication du fichier à ouvrir
- sortie : RETCODE, RFIL
- fonction : ouvrir logiquement et physiquement un fichier correspondant à l'opération demandée

Initialement deux fichiers sont disponibles en lecture et deux en écriture. Suivant l'opération demandée, il faut voir si un des deux fichiers potentiels est libre. Si c'est le cas, il sera ouvert et sa référence sera fournie dans RFIL. Sinon [ERREUR 26].

Le nom-système du fichier est contenu dans FILNAME. L'ouverture en lecture consiste à faire une opération [RESET]. Si le fichier n'existe pas l'erreur est signalée [ERREUR 25];
l'ouverture en écriture génère un nouveau fichier physique.

- messages d'erreur :

25. Le fichier tel qu'il a été spécifié n'est pas connu du système.
26. Deux fichiers au maximum peuvent être ouverts simultanément dans un mode donné.

*** F_FERMETURE ***

- entrée : REFILE = référence correcte d'un fichier de la BD
- fonction : si le fichier est ouvert, le fermer logiquement et physiquement, en prenant soin de ne pas perdre le contenu de la zone tampon (BUFFER)

*** ART_ARG ***

- entrée : REFILE = référence valide d'un fichier de la BD, BUFFER (REFILE), CURRENT (REFILE), ARG_PTR, TAILLE
- sortie : RFIELD, ARG_PTR, CURRENT (REFILE)
- fonction : transférer TAILLE octets du BUFFER (REFILE), à partir de la position CURRENT (REFILE), dans RFIELD, à partir partir de la position ARG_PTR.

*** ARG_ART ***

- entrée : REFILE = référence valide d'un fichier de la BD, Z_VALIT, CURRENT (REFILE), ARG_PTR, TAILLE
- sortie : BUFFER (REFILE), ARG_PTR, CURRENT (REFILE)
- fonction : transférer TAILLE octets de Z_VALIT, à partir de la position ARG_PTR, dans BUFFER (REFILE), à partir de la position CURRENT (REFILE).

*** LECTURE_FICHIER ***

- entrée : REFILE = référence valide d'un fichier de la BD
- sortie : BUFFER (REFILE), valeur booléenne
- fonction : lire dans BUFFER (REFILE) le bloc d'octets suivant du fichier de référence REFILE; indiquer si cette lecture a fourni un résultat (pas la fin du fichier)

*** ECRITURE_FICHIER ***

- entrée : REFILE = référence valide d'un fichier de la BD BUFFER (REFILE)
- sortie : valeur booléenne
- fonction : écrire à la fin du fichier de référence REFILE le bloc d'octets contenu dans BUFFER (REFILE); indiquer si cette opération s'est bien déroulée (place sur le disque).

6.4. SPECIFICATIONS DES PRIMITIVES.

*** OUVERTURE DE LA BASE DE DONNEES ***

- entrée : COP = 11, SSNAME
- sortie : RETCODE
- réalisation :
  - si SSNAME <> "BDFI" alors [ERREUR 4]
  - [ajouter 1 à FI_NOUVBD]
- messages d'erreurs :

  4. La Base de Données est inconnue

*** FERMETURE DE LA BASE DE DONNEES ***

- entrée : COP = 12
- sortie : RETCODE
- réalisation :
  - si la Base de Données n'est pas ouverte alors [ERREUR 7]
    [FI_NOUVBD = 0] ?
  - [soustraire 1 de FI_NOUVBD]
  - si la Base de Données est totalment fermée
    [FI_NOUVBD = 0] ?
    alors fermer les fichiers que l'utilisateur a omis de fermer
    [F_FERMETURE]
- messages d'erreur :

  7. La Base de Données n'est pas ouverte

*** OUVERTURE D'UN FICHIER ***

- entrée : COP = 22, FILNAME, PROTECT
- sortie : RETCODE, RFIL
- réalisation :
  - si la Base de Données n'est pas ouverte alors [ERREUR 7]
    [FI_NOUVBD = 0] ?
  - si le mode d'ouverture souhaité est la lecture
    [PROTECT = 2 ou 6] ?
    alors demander l'ouverture du fichier FIN1 ou FIN2
    [F_OUVERTURE (lect)]

- si le mode d'ouverture voulu est l'écriture
[PROTECT = 3 ou 7] ?
alors demander l'ouverture du fichier FOUT1 ou FOUT2
[F_OUVERTURE (écr)]

sinon [ERREUR 19]

- messages d'erreur :

7. La Base de Données n'est pas ouverte
19. PROTECT doit être égal à 2,6 (lecture ou 3,7 (écriture

*** FERMETURE D'UN FICHIER ***

- entrée : COP = 24, RFIL

- sortie : RETCODE

- réalisation :

- si la Base de Données n'est pas ouverte alors [ERREUR 7]
[FI_NOUVBD = 0] ?
- si RFIL n'est pas correct alors [ERREUR 9]
[RFIL = 1 ou 2 ou 3 ou 4] ?

demander la fermeture du fichier [F_FERMETURE]

- messages d'erreur :

7. La Base de Données n'est pas ouverte
9. RFIL n'est pas la référence d'un fichier de la BD

*** ACCES SEQUENTIEL AUX ARTICLES D'UN FICHIER ***

- entrée : COP = 33, RFIL, COREC

- sortie : RETCODE, RFIELD

- réalisation :

- si la Base de Données n'est pas ouverte alors [ERREUR 7]
[FI_NOUVBD = 0] ?
- si RFIL n'est pas valide alors [ERREUR 20]
[RFIL <> 1 et 2] ?
- si le fichier n'est pas ouvert alors [ERREUR 13]
[FI_ETAT (RFIL) = 0] ?

lire dans le BUFFER (REFILE) un nombre déterminé d'octets
(COREC se situe dans l'intervalle [1..255])
ou lire jusqu'à un symbole de fin de ligne (max. 255 octets)
(COREC vaut 0)

si nécessaire, faire une lecture physique dans le fichier
[LECTURE_FICHIER]

- messages d'erreur :

  7. La Base de Données n'est pas ouverte
  13. Le fichier n'est pas ouvert
  20. Le fichier doit être ouvert en lecture

*** CREATION D'UN ARTICLE ***

- entrée : COP = 61, RFIL, COREC, Z_VALIT

- sortie : RETCODE, RFIELD

- réalisation :

  - si la Base de Données n'est pas ouverte alors [ERREUR 7]
    [FI_NOUVBD = 0] ?
  - si RFIL n'est pas valide alors [ERREUR 21]
    [RFIL <> 3 et 4] ?
  - si le fichier n'est pas ouvert alors [ERREUR 13]
    [FI_ETAT (RFIL) = 0] ?

  écrire dans le BUFFER (REFILE) un nombre déterminé d'octets
  (COREC se situe dans l'intervalle [1..255])
  ou écrire jusqu'à un symbole de fin de ligne (max. 255 octets)
  (COREC vaut 0)

  si nécessaire, faire une écriture physique dans le fichier
  [ECRITURE_FICHIER]

- messages d'erreur :

  7. La Base de Données n'est pas ouverte
  13. Le fichier n'est pas ouvert
  21. Le fichier doit être ouvert en écriture

A N N E X E III

IMPLEMENTATION DU SYSTEME IDML PARTAGE

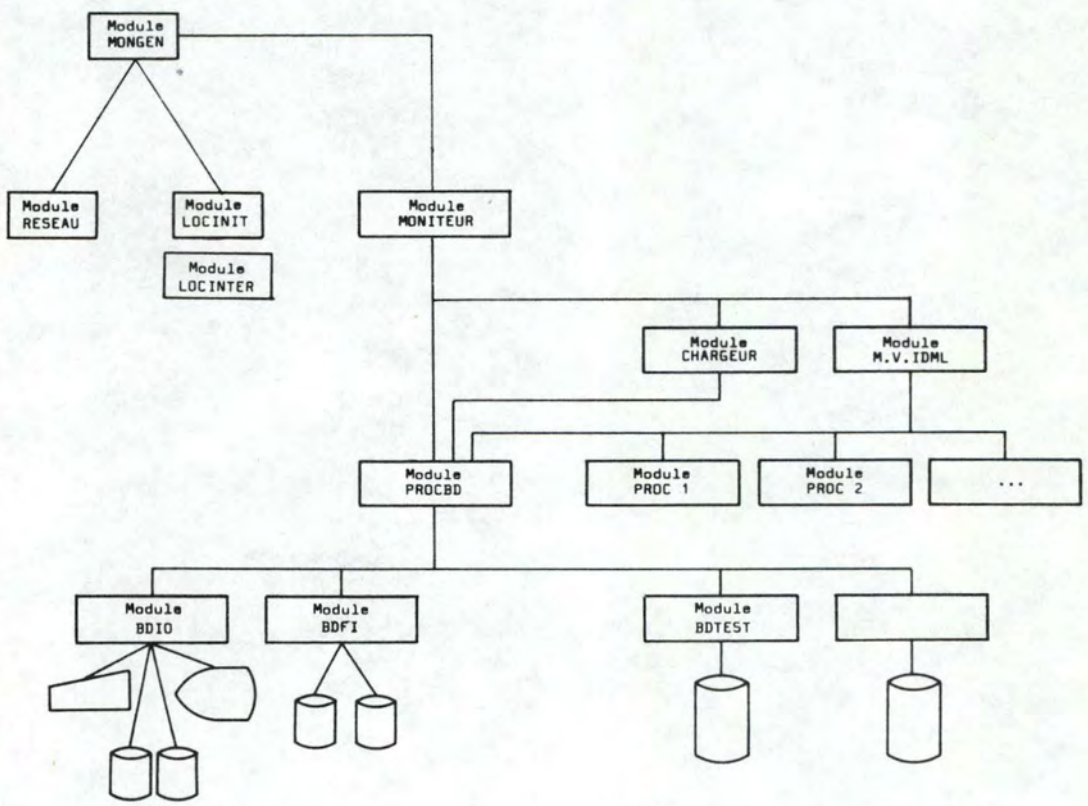
SPECIFICATIONS DES DIFFERENTS MODULES

1. RAPPEL DE L'ARCHITECTURE.

Le système réalisé permet à trois utilisateurs de travailler simultanément, au niveau Primitives ou au niveau Langage de base.

Un des utilisateurs est l'utilisateur local tandis que les deux autres sont sur le réseau, et travaillent avec la système IDML par l'intermédiaire d'un programme d'application particulier.

Voici l'architecture globale (niveau Primitives et niveau Lgbs) du système IDML partagé .



Pour chacun des modules qui la constitue, nous spécifierons la fonction globale qu'il permet de réaliser. Nous définirons les variables spécifiques qu'il utilise et nous analyserons les procédures principales.

Les procédures en Pascal MT+ se trouvent dans l'annexe IV.

- Chap. 2. le système complet : SIDMLS

  - Shared Interactive Data Manipulation -
  - Language System -

- Chap. 3. le moniteur général : MONGEN
- Chap. 4. le réseau : RESEAU, NETPAS, NETMAC
- Chap. 5. l'utilisateur local : LOCINTER, LOCINIT
- Chap. 6. le moniteur : MONITEUR
- Chap. 7. le chargeur : CHARGEUR
- Chap. 8. la machine virtuelle : MVIDML
- Chap. 9. le processeur Base de Données : PROCBD
- Chap. 10. les processeurs spécialisés : PROC1, PROC2, PROC3, PROC5

Quant aux interfaces Base de Données (conventionnelle BDTEST, Terminal BDIO et Fichiers Standard BDFI), leur description fait l'objet de l'annexe II (Chapitres 4, 5 et 6).

Prim. : abréviation utilisée pour niveau Primitives;
Lgbs. : abréviation utilisée pour niveau Langage de base.

2. LE SYSTEME COMPLET : SIDMLS.

2.1. Fonction.

Le programme SIDMLS met en oeuvre les concepts du système IDML partagé sur un réseau de micro-ordinateurs North Star.

Il supporte actuellement un utilisateur local et deux utilisateurs réseaux.

Ce module principal a pour seul but de lancer l'exécution du système, c'est-à-dire le Moniteur Général.

L'arrêt du programme est sous le contrôle de l'utilisateur local.

3. LE MONITEUR GENERAL : MONGEN.

3.1. Fonction.

En tant que moniteur du système multi-utilisateurs en temps réel, le Moniteur Général se doit d'allouer l'unité centrale successivement à chaque utilisateur, et ce pour la durée d'une étape. Chaque étape du processus (session-utilisateur) est constituée d"une action logique dans cette session.

3.2. Informations.

Trois informations sont primordiales :

- USER :

  à tout instant, cette variable contient la référence interne de l'utilisateur qui occupe l'unité centrale.

  L'utilisateur local a la référence 0 tandis que les utilisateurs réseaux reçoivent les références 1 et 2.

  Les références des sites réseaux sont en correspondance avec l'adresse physique des sites sur le réseau.
  L'adresse 97 est réservée au système IDML partagé. Les adresses suivantes sont celles des stations sur le réseau. Actuellement, le nombre des stations est limité à deux, ce qui rend uniquement les adresses 98 et 99 valides. voir module RESEAU

  Le roulement des utilisateurs pour l'occupation de l'unité centrale se fait sur base de cette référence.
  C'est cette même référence qui accompagnera l'utilisateur dans tout le système, seul le module RESEAU devant établir la correspondance entre l'adresse physique et la référence interne d'un utilisateur.

- SESSION :

  cette variable indique, pour chaque utilisateur potentiel l'état de son travail.

  Plusieurs renseignements sont disponibles :

  - ETAPE = prochaine étape à exécuter;
  - TYPE-OF-WORK = niveau de langage dans lequel il travaille (Prim, Lgbs);
  - FIN-TRAIT = indicateur de fin de traitement qui peut être positionné à différents endroits du système;

- FIN-EXEC-MONGEN :

  positionnè par l'utilisateur local, ce drapeau commande l'arrêt du programme. voir module LOCINIT

3.3. Les procèdures principales.

[Les arbres programmatiques qui y correspondent ont : ètè exposès dans la partie principale aux points 3.3.3. et 3.4.3]

- *** procèdure MONGEN ***

  - initialiser le système;
  - allouer l'unitè centrale successivement à chaque utilisateur potentiel;
  - clôturer le système.

- *** procèdure TRAV-LOCAL ***

  Suivant l'ètape à exècuter :

  0. initialiser le dialogue à l'écran " Please Start : " et positionner l'indicateur d'ètape sur 1.

  1. analyser le message èventuel introduit par l'utilisateur :

     S'il s'agit du message " Start ", rèaliser le dialogue et positionner à l'ètape 2;

     s'il s'agit du message " End ", l'indicateur de fin de programme est positionnè.

  2. exècuter le travail dont le type a ètè dèfini lors du dialogue. [EXECUTION]

     Si l'indicateur de fin de traitement est positionnè, l'ètape suivante à exècuter sera l'ètape 3, sinon on reprendra la même ètape.

  3. terminer le dialogue , puis rèinitialiser le travail et enclencher l'ètape 0.

- *** procèdure TRAV-RESEAU ***

  Suivant l'ètape à exècuter :

  0. analyser le message èventuel envoyè par l'utilisateur.

s'il s'agit d'une demande de communication au niveau Prim, se positionner à l'étape 1;

s'il s'agit d'une demande de communication au niveau Lgbs, recevoir les données constituant le programme en code exécutable et engager l'étape 10;

1. 10. exécuter le travail demandé [EXECUTION]

si le travail est au niveau Prim. et si l'indicateur de fin de traitement est positionné, l'étape suivante sera 2, sinon elle restera 1;

si le travail est au niveau Lgbs, on passera à l'étape 11.

11. communiquer à l'utilisateur les résultats du travail en Lgbs.

Si la communication a lieu, on passera à l'étape 2 sinon on recommencera cette étape 11

2. réinitialiser le travail et enclencher l'étape 0.

- *** procédure EXECUTION ***

Si l'utilisateur est l'utilisateur local :

Suivant le type de travail :

- Prim : lecture des paramètres, appel du processeur BD et écriture des résultats

- Lgbs : appel du Moniteur et positionnement de la fin de traitement

Si l'utilisateur est un utilisateur réseau :

Suivant le type de travail :

- Prim : lecture des paramètres; si la communication a eu lieu, appel du processeur BD et écriture des résultats

- Lgbs : appel du Moniteur et positionnement de la fin de traitement

4. LE TRAVAIL AVEC LE RESEAU : RESEAU, NETMAC, NETPAS

4.1. Fonction.

Il prend en charge tous les traitements directement liés au caractère " réseau " de l'utilisateur :

- liaison avec le réseau et son logiciel de base;
- mise en application du protocole de haut-niveau;
- gestion des messages en attente;
- traitement lié à la session Primitives;
- traitement lié à la session Langage de base.

Le logiciel de base est stocké dans deux modules particuliers :

- le module NETMAC qui comprend les procédures de bas niveau écrites en Assembleur;
- le module NETPAS qui sert d'interface entre ce bas niveau et le programme Pascal.

4.2. Informations.

La communication sur le réseau se fait par l'intermédiaire de messages des octets maximum. Un mécanisme de " panier de réception " est fourni par le logiciel de base. L'extraction d'un message se fait dans l'ordre FIFO.

Vu que le programme demandera le message d'un utilisateur particulier, il faut améliorer cette gestion du panier. Ce qui est fait dans le programme lui-même. Le panier de réception peut être vidé dans des buffers associés à chaque utilisateur. Une zone de la taille de deux messages réseau est prévue à cet effet.

- NET-MSG :

  message servant de liaison entre le réseau et le programme

- NET-MSG-LEN :

  longueur effective du message NET-MSG (en nombre d'octets)

- BUF-NET :

  buffers du réseau associé à chaque utilisateur.

  Cette zone peut contenir deux messages.

- BUF-NET-LEN :

associée à chaque BUF-NET, cette variable indique la longueur effective du message contenu dans le buffer.

La longueur 0 signifie l'absence de message.

- COMMUNICATION :

variable indiquant si les données demandées sont disponibles ou non.

Cet indicateur est utilisé pour donner le résultat de la lecture des paramètres d'une primitive ou de la réception d'un programme en code exécutable.

- MY-ADDR :

contient l'adresse-réseau du site sur lequel est implanté ce système.

Le choix de l'adresse est laissé au processus qui tourne sur la station. Ce n'est donc pas le réseau qui impose cette adresse. Il vérifie seulement que l'adresse n'est déja pas utilisée par un autre site.

Nous avons fixé cette adresse à la valeur 97.

4.3. Format des informations.

- demande de communication :
  - byte 1 : type du message = DMDE-COM
  - byte 2 : type de travail = PRIM ou LGBS

- fin de communication, acknoledgement ou non-acknoledgement :
  - byte 1 : type du message = FIN-COM ou ACK ou NO-ACK

- requête ou réponse Primitive :

Premier message :

  - byte 1 : type du message = REQ-PRIM ou REP-PRIM
  - byte 2 : indication de chaînage (la suite des données est dans un second message)
  - byte 3 à byte 7 : adresses relatives dans la zone des données des cinq paramètres
  - byte 8 : indication du dernier byte occupé dans le dernier message qui constitue la primitive
  - byte 9 à byte 250 : données

Second message éventuel :

- byte 1 : type du message = REQ-PRIM ou REP-PRIM
- byte 2 : indication de chaînage = 0
- byte 3 à byte 250 : données

- Bloc de caractères (programme ou résultat-fichier ou résultat-terminal)

  - byte 1 : type du message = BC-LGBS ou BC-RES-FI ou BC-RES-IO
  - byte 2 : numéro d'ordre du message (non utilisé pour le moment)
  - byte 3 à byte 250 : données

4.4. Les procédures du logiciel de base.

- *** procédure OPEN-LINK ***

  - entrée : MY-ADDR
  - sortie : CODE-ERR
  - fonction : demande d'initialisation d'une liaison virtuelle.

    Cette procédure permet au processus d'application d'accéder au réseau avec pour adresse MY-ADDR. Elle renvoie un CODE-ERR éventuel :

    0. pas d'erreur : la liaison est ouverte.
    1. la liaison est déja ouverte (elle reste ouverte)
    2. une autre station est déja connectée au réseau sous la même adresse (la liaison n'est pas ouverte)
    3. erreurs répétées lors de l'accès au bus (la liaison n'est pas ouverte)

- *** procédure CLOSE-LINK ***

  - sortie : CODE-ERR
  - fonction : demande de clôture.

    Cette procédure a pour but de fermer l'accès au réseau.
    Après l'appel à cette procédure, la station est déconnectée du réseau, de sorte que la station ne puisse plus ni recevoir ni émettre des messages sur le réseau. Un CODE-ERR est retourné au programme :

    0. pas d'erreur : la liaison est fermée

1. la liaison était déja fermée, ou pas ouverte (la liaison reste fermée)

2. la liaison est fermée mais des messages sont encore disponibles dans le panier de réception

- *** procédure SEND ***

  - entrée : TO-ADDR, NET-MSG-LEN, NET-MSG

  - sortie : CODE-ERR

  - fonction : demande d'envoi d'un message

    Cette procédure a pour but d'envoyer un message NET-MSG dont on connait la longueur NET-MSG-LEN à la station dont on spécifie l'adresse TO-ADDR.
    Pour des raisons diverses, il se peut que le destinataire n'ait pas reçu le message ou l'ait refusé. Pour savoir dans quelles conditions s'est effectué cet envoi, un CODE-ERR est renvoyé :

    0. pas d'erreur : le message a été accepté par la station destinatrice

    1. trop d'essais-collisions; le message est entré plusieurs fois en collision avec un autre message

    2. trop d'essais-parasites; à plusieurs reprises, le message a été mal reçu par la station destinatrice

    3. trop d'essais-station inexistante; le message a été envoyé plusieurs fois mais aucun acquittement en provenance de la station destinatrice n'a été reçu. Cette station semble absente.

    4. station destinatrice pas prête: le message a été convenablement reçu par la station destinatrice mais celle-ci ne dispose plus d'assez de place pour stocker ce message

    5. erreur inconnue : une erreur inconnue est apparue en cours de transmission

    6. adresse de destination invalide: 0 ou l'adresse de la station elle-même

    7. longueur du message invalide : 0 ou supérieure à 250

    8. accès au réseau non autorisé

- *** procédure RECEIVE ***

  - entrée : WAIT

  - sortie : FROM-ADDR, NET-MSG-LEN, NET-MSG, CODE-ERR

  - fonction : demande de réception d'un message

Deux solutions sont possibles lors de l'appel de cette procédure.
Dans la première solution (WAIT = vrai), le processus d'application est bloqué jusqu'à disponibilité d'un message;
dans le second cas (WAIT = faux), si aucun message n'est disponible, le processus n'est pas bloqué. Si un message est disponible, il est fourni au processus d'application.

On peut encore faire appel à cette procédure après la fermeture de l'accès au réseau pour prendre les messages restant qui ont été reçus avant cette clôture.

0. pas d'erreur : un message est renvoyé dans NET-MSG

1. pas de message disponible (option WAIT = faux)

2. pas de message disponible (accès au réseau fermé

4.5. Les procédures principales.

- *** fonction CORR-ADDR ***

  Recevant la référence interne d'un utilisateur, cette fonction renvoie l'adresse physique de la station sur le réseau qui lui correspond.

- *** fonction CORR-REF ***

  Recevant l'adresse physique d'une station sur le réseau, cette fonction renvoie la référence interne de l'utilisateur qui lui correspond.

- *** procédure NET-OPEN ***

  Appel à la procédure de base OPEN-LINK

- *** procédure NET-WRITE ***

  Connaissant la référence interne du correspondant et le type de message à lui envoyer, cette procédure se charge de mettre au point le message contenu dans NET-MSG et d'appeler la procédure de base SEND.

  Si après cinq essais, le message n'est pas arrivé correctement à destination, la procédure est stoppée et un code d'erreur est retourné.

- ***procédure NET-READ ***

  Cette procédure consiste à vider le panier de réception jusqu'à ce que l'on ait trouvé un message du correspondant demandé ou que le panier soit vide . Il s'agit donc d'appeler une ou plusieurs fois la procédure de base RECEIVE avec WAIT=faux

- *** procédure NET-CLOSE ***

  Cette procédure demande la fermeture de l'accès au réseau CLOSE-LINK. Si des messages sont encore en attente dans le panier de réception , un message de non-acknoledgement est envoyé aux sites émetteurs de ces messages.

- *** fonction RCV-MESS ***

  Cette fonction consiste à signaler la disponibilité vis-à-vis du programme d'un message particulier pour un utilisateur donné.

  Il peut s'agir d'un message DMDE-COMM ou ACK ou REQ-PRIM ou REQ_LGBS. Un message autre que celui demandé est signalé par une erreur et un message de non-acknoledgement est immédiatement envoyé à l'utilisateur.

  Une remarque concernant le message de fin de communication : sa réception ne constitue pas une erreur à proprement parler. Suivant le contexte, il pourra être compris soit comme une indication de fin de transfert soit comme une indication de fin de traitement.

- *** procédure RD-NET-PRIM ***

  - voir si une requête-primitive est disponible;
  - si c'est le cas, préparer les paramètres pour le processeur BD.

- *** procédure WR-NET-PRIM ***

  - calculer la longueur effective de la réponse sur base des informations contenues dans le premier message REQ-PRIM qui lui correspond;
  - demander l'émission de un ou deux messages à l'utilisateur concerné

- *** procédure RD-NET-LGBS ***

  - préparer le nom du fichier dans lequel on va stocker les données : il s'agit d'un nom de format standard
    FEXEC<adr.réseau util>.E

- réceptionner les messages REQ-LGBS et les enregistrer dans le fichier

- créer le fichier de commandes FIDML.IN (comprenant une demande d'exécution du programme puis un ordre de terminaison)

- *** procédure WR-REP-LGBS ***

  - préparer le nom du fichier de résultats : DTCOM<adr.réseau util>

  - si ce fichier existe, l'envoyer à l'utilisateur sous forme de messages BC-RES-FI;

  - si ce fichier n'existe pas, lui envoyer le fichier batch (copie de l'écran). Les messages portent l'indication BC-RES-IO

5. LE TRAVAIL AVEC L'UTILISATEUR LOCAL : LOCINIT, LOCINTER

5.1. Fonction.

Le module LOCINIT a pour but d'initialiser une session de l'utilisateur local.

Le module LOCINTER prend en charge tout le travail interactif avec cet utilisateur. Ce module n'a pas un caractère général. Il a été conçu pour permettre à l'utilisateur de travailler sur la Base de Données BDTEST. Toute la gestion d'écran est donc définie par rapport à cette interface.

- dialogue permettant de connaître le niveau de travail et l'emploi éventuel d'un fichier trace si l'on travaille par primitives.
- lecture des paramètres constituant une primitive
- écriture du résultat de l'exécution d'une primitive

5.2. Informations.

- LINE-SCREEN, COLUMN-SCREEN :

  paramètres de la gestion d'écran

- Z-CODES :

  tableau de 30 octets pour contenir les paramètres Z-CODES

- Z-IDENT :

  tableau de 90 positions permettant de stocker les paramètres Z-IDENT

- Z-VALUES :

  tableau de 128 octets utilisées comme zone Z-VALIT et RFIELD

  Ces trois tableaux constituent les zones de paramètres qui sont l'interface entre l'utilisateur local et le processeur Base de Données.
  La taille de la zone Z-VALUES a été limitée à 128 caractères car aucun article de la Base de Données BDTEST n'excède cette taille.

- FILE-RES-PRIM :

  fichier de texte permettant de garder une trace de la session de travail. Pour chaque primitive envoyée, une annotation est inscrite sur ce fichier indiquant quelle opération a été demandée et quel en est le résultat.

5.3. Les procédures principales.

La procédure de lecture du clavier offerte par le Pascal MT+ est une procédure synchrone, avec attente. Vu que l'on ne sait pas si l'utilisateur local va travailler, il a fallu écrire un petit logiciel permettant de savoir si l'utilisateur a tapé des caractères et si oui, de les traiter en conséquence.

Il s'agit des procédures RDCLAV (écrite en Assembleur), AFFICHER-CHAR, MEMORISER-CHAR et DELETE-CHAR.

- *** fonction TRAIT-MESS-CORRECT ***

  Cette fonction vérifie si un message entier (terminé par un <LINE-FEED>) a été écrit par l'utilisateur. Si ce n'est pas le cas, la partie éventuellement déja inscrite est mémorisée. Si le message est terminé, son contenu est vérifié.

  Le message "START" signifie que l'utilisateur veut commencer une session;
  le message "END" signifie l'arrêt du programme.
  Tout autre message est ignoré.

- *** procédure OPEN-DIALOGUE ***

  - demande du niveau de travail : PRIM ou LGBS
  - dans le cas du travail Primitives, voir si l'utilisateur désire un fichier trace; et si oui quel en est le nom sur disque.

- *** procédure RD-REQ-PRIM ***

  - affichage du cop courant et lecture du cop suivant
  - en fonction du cop, affichage d'une grille reprenant les différents arguments nécessaires pour constituer la primitive de cop donné.De plus les valeurs courantes de ces arguments sont également affichées.
  - lecture des nouvelles valeurs des arguments, avec possibilité de s'aider d'un petit lexique pour rappeler la signification des arguments
  - préparation des adresses des arguments (en fonction partiellement du cop)

L'entrée des valeurs se fait comme suit :

- le curseur se positionne devant la zone à remplir
- si l'utilisateur veut garder la valeur courante, il appuye sur la touche <LINE-FEED>
- si l'utilisateur veut faire appel au lexique ,il tape le caractère H (Help).Le message d'explication apparaît au bas de l'écran et le curseur reste positionné devant la zone de valeur.
- si l'utilisateur veut introduire une nouvelle valeur, il se positionne au début du champ en appuyant sur la barre d'espacement (ou tout autre caractère non repris ci-avant, mais qui ne sera pas enregistré comme premier caractère de la valeur). La fin de la valeur est signalée par le caractère <RETURN>.
- le curseur est positionné devant le champ suivant.

- *** procédure WR-REP-PRIM ***

  - écriture du résultat sur l'écran. Tous les paramètres modifiés par l'interface sont visualisés
  - si un fichier trace a été ouvert, quelques informations sur l'opération y sont inscrites.
  - mise-à-jour éventuelle de l'indicateur de fin de traitement : l'utilisateur doit indiquer s'il souhaite continuer sa session ou s'il a terminé.

- *** procédure CLOSE-DIALOGUE ***

  Cette procédure a pour rôle de fermer le fichier de résultats-primitives, si celui-ci a été ouvert préalablement

6. LE MONITEUR DU SYSTEME IDML : MONITEUR.

6.1. Fonction.

Le rôle du moniteur est de diriger et de contrôler le déroulement des différentes phases du traitement d'un programme. Il supervise les modules de compilation, reliage, chargement et exécution.

Les commandes à exécuter lui sont fournies par l'utilisateur via le terminal ou ce qui tient lieu de terminal, c'est-à-dire le fichier batch.

La syntaxe des commandes disponibles est limitée :

- un ensemble de commandes pour l'édition;
- COMP (compilation du programme courant);
- LINK (assemblage du programme objet courant);
- LOAD (chargement du programme exécutable courant);
- <nom-fichier> (exécution de la version la plus récente);
- <nom-fichier>.<extension> (exécution de la version la plus récente)

Le rôle du moniteur peut dès lors se résumer à lire une commande, l'analyser pour enclencher le(s) module(s) correspondant(s), et ainsi de suite jusque quand il reçoit l'ordre de terminaison END.

6.2. Informations.

Peu d'informations sont nécessaires au moniteur pour réaliser sa tâche vu que les données sur lesquelles il travaille arrivent au coup par coup.

- Z-CODES :

  tableau de 30 octets pour contenir les paramètres Z-CODES

- Z-IDENT :

  tableau de 90 positions pour stocker les paramètres d'identification

- Z-VALUES :

  tableau de 128 caractères utilisé comme zone d'émission et de réception de valeurs d'articles.

Ces trois zones permettent au moniteur de travailler interactivement avec le terminal (ou ce qui tient lieu de terminal) comme sur une Base de Donnees. Elles respectent dès lors les formats spécifiés pour les interfaces Base de Donnees.

- SREF-IO :

  référence de la Base de Donnees Terminal

- RFIL-IN :

  référence du terminal en entrée (clavier ou fichier batch-in)

- RFIL-OUT :

  référence du terminal en sortie (écran ou fichier batch-out)

6.3. Les procédures principales.

- *** fonction OPEN-BDIO ***

  demander l'ouverture de la Base de Donnees Terminal; renvoyer le résultat de cette opération et mémoriser la référence de la BD.

- *** fonction OPEN-TERM ***

  demander l'ouverture du clavier et de l'écran; renvoyer le résultat de cette opération et mémoriser la référence de chacun de ces "fichiers".

- *** fonction READ-REC ***

  demander la lecture d'un article sur le clavier; la longueur de l'article est déterminée par la présence d'un caractère <RETURN>; renvoyer le résultat de cette opération.

- *** fonction WRITE-REC ***

  demander l'écriture d'un article sur l'écran; la longueur de l'article est passée comme paramètre de cette routine; renvoyer le résultat de cette opération.

- *** fonction CLOSE-TERM ***

  demander la fermeture du terminal (clavier et écran); renvoyer le résultat de cette opération.

- *** fonction CLOSE-BDIO ***

  demander la fermeture de la Base de Bonnées Terminal; renvoyer le résultat de cette opération.

Pour toutes les opérations que nous venons de citer, la zone de transmission des données et des résultats est constituée des tableaux Z-CODES, Z-IDENT et Z-VALUES. La demande d'exécution de chacune des primitives est adressée au processeur Base de Données et s'adresse à l'interface Base de Données Terminal.

- *** procédure LOAD-RUN ***

  - lancer le module CHARGEUR et lui fournir le nom du fichier potentiel à charger;
  - s'il y a une erreur, la signaler et arrêter le travail
  - s'il n'y a pas d'erreur, avertir l'utilisateur du bon chargement puis initialiser la machine virtuelle IDML
  - s'il y a un problème fatal, le signaler à l'utilisateur; sinon l'avertir de la terminaison du travail

- *** procédure MONITEUR ***

  - réaliser quelques opérations d'initialisations (ouverture du terminal);
  - si ces opérations se sont déroulées correctement, lire une commande et l'analyser.

    si la commande peut être réalisée (chargement et exécution d'un programme contenu dans un fichier potentiel), superviser la réalisation de la demande (via la procédure LOAD-RUN)

    recommencer cette manoeuvre jusqu'à la commande END.
  - réaliser quelques opérations de clôture (fermeture du terminal)

  Des erreurs peuvent être signalées à différents niveaux :

  - communication avec la Base de Données Terminal
  - commande erronée ou non implémentée
  - erreur signalée par le chargeur ou l'interprète

  Chacune de ces erreurs fera, si possible, l'objet d'un message à l'écran de l'utilisateur, qui pourra alors introduire une nouvelle commande.

7. LE CHARGEUR DU SYSTEME IDML : CHARGEUR.

7.1. Fonction.

Le chargeur est l'élément du système qui charge dans la mémoire de la machine virtuelle un programme en code interne exécutable.

Il consiste à ranger aux adresses absolues prévues lors de l'assemblage les instructions, les descripteurs et le valeurs initiales. Ce programme en code interne est stocké dans un fichier sous forme de segments.

Suite au chargement, le chargeur fournit dans le PC l'adresse de la première instruction à exécuter et dans le CDBA l'adresse absolue d'implantation du bloc des descripteurs.

Un fichier contenant un programme en code exécutable doit porter un nom dont l'extension est .E
Ce fichier contient en outre un descripteur général (type du programme, numéro de version, nombre de segments).

Quatres types de segments ont été distingués :

- instructions
- descripteurs
- valeurs initiales
- zone de travail libre

7.2. Informations.

Deux types d'informations sont nécessaires pour que le chargeur puisse s'exécuter correctement :

- les informations liées au fichier qui contient le programme à charger.

  Ce fichier est sous le contrôle de l'interface BD Fichiers Standard.

- les informations liées à la mémoire de la machine virtuelle.

- NOMFI :

  nom du fichier contenant le programme en code exéecutable.

  Ce nom est fourni au chargeur par le moniteur.

- Z-CODES .

- Z-IDENT .

- Z-VALUES .

- MEMOIRE :

  la mémoire de la machine virtuelle est un ensemble de 4000 octets

  En ce qui concerne les instructions et les descripteurs, la mémoire est vue comme une suite d'élèments de 4 octets (mot standard = 32 bits) tandis que le mot standard pour les valeurs est l'octet.

  Nous considèrerons que les adresses d'implantation prèvues par l'assemblage et contenues dans les descripteurs de segments sont exprimées en terme de mots de 32 bits, afin d'uniformiser ces adresses.

- PC :

  compteur ordinal de la machine virtuelle

- CDBA :

  adresse absolue d'implantation en mémoire du bloc de descripteurs des valeurs de la procédure courante.

Actuellement, les programmes pouvant s'exécuter sont constitués d'une procédure unique qui est la procédure principale.

7.3. Format des informations.

- Descripteur général du fichier :

  - byte 1 : type du fichier = (FI-SCE, FI-OBJET ou) FI-EXEC
  - byte 2 : numéro de version
  - bytes 3 et 4 : adresse absolue d'implantation de la première instruction à exécuter
  - byte 5 : nombre de segments à analyser

- Descripteur d'un segment :

  - byte 1 : type du segment = S-INST, S-DESC, S-VALINIT ou S-LIBRE
  - bytes 2 et 3 : adresse d'implantation du segment en mémoire; en considérant la mémoire comme un ensemble de 1000 mots de 32 bits

- bytes 4 et 5 : nombre de mots à transférer.
  la taille du mot considéré est 32 ou 8 bits suivant le type du segment
- bytes 6 et 7 : nombre de mots à réserver (non utilisé ici)
- byte 8 : (non utilisé)

7.4. Les procédures principales.

- *** fonction READ-REC ***

  demande de lecture de NB-BYTES dans le fichier; cette requête s'adresse à l'interface BD Fichiers Standard;

- *** procédure TRAIT-SEGMENT ***

  - lecture du descripteur du segment suivant
  - si ce descripteur existe, l'analyser et en tirer les informations relatives à l'adresse d'implantation et à la taille des données à transférer.

    s'il s'agit d'un segment de descripteurs de valeurs, vérifier si l'adresse d'implantation ne doit pas être fournie au CDBA.
  - effectuer le transfert : lecture dans le fichier, puis chargement dans la MEMOIRE à l'adresse prévue.

    Si la lecture ne s'est pas bien réalisée (READ-REC=faux), arrêter le transfert.

- *** procédure CHARGEUR ***

  - demande d'ouverture de la Base de Données Fichiers Standard
  - demande d'ouverture en lecture du fichier NOMFI. Ce nom a été passé comme paramètre par le Moniteur.
  - si ces deux opérations se sont bien déroulées, demande de lecture du descripteur général;

    vérification du type du fichier et enregistrement du nombre de segments à traiter.
  - pour le nombre de segments enregistré, effectuer le TRAIT-SEGMENT.

    Si la fin du fichier est atteinte avant que le nombre de segments enregistré ait été traité, l'erreur est signalée.
  - demande de fermeture du fichier
  - demande de fermeture de la Base de Données Fichiers Standard

  Toute erreur entraînant l'arrêt du chargeur est signalée au Moniteur

8. LA MACHINE VIRTUELLE IDML : M.V.IDML

8.1. Fonction.

La machine virtuelle du système IDML exècute des programmes qui ont été prèalablement chargès dans sa mèmoire.

Elle a à sa disposition deux registres particuliers : le PC et le CDBA.

Elle est implèmentèe comme un interprète du langage interne.

8.2. Informations.

La source d'information sur laquelle travaille la M.V.IDML est èvidemment sa mèmoire.

- MEMOIRE :

  la mèmoire est un ensemble de 4000 octets ou 1000 mots standard IDML (32 bits).

  L'interprètation et la structuration des èlèments qui y sont rangès est à charge de la M.V.IDML.

- PC :

  compteur ordinal; contient l'adresse de l'instruction courante

- CBDA :

  adresse du bloc courant des descripteurs des valeurs

- FIN-INTERPR :

  indicateur de la fin du travail de la machine virtuelle.

  Cet indicateur peut être positionnè soit par la rencontre de l'instruction de fin d'exècution STOP, soit parce qu'une erreur fatale s'est produite.

8.3. Format des informations.

Le format des instructions et des descripteurs n'est par repris ici.
Il a fait l'objet du Chapitre 2.3. dans la partie principale.

8.4. Les procédures principales.

- *** procédure PC-SUIVANT ***

  Faire pointer le PC vers l'instruction suivante à exécuter, qui devient l'instruction courante.

  L'adresse de cette instruction est calculée sur base de la valeur courante du PC augmentée de NB-MOTS-STD. NB-MOTS-STD peut contenir une valeur négative.

  Une erreur est signalée si l'adresse dépasse les dimensions de la mémoire.

- *** procédure DESCR-SUIVANT ***

  Connaissant l'adresse d'un descripteur de valeur cette procédure a pour but de donner l'adresse du descripteur suivant dans la mémoire.

  Cette adresse est calculée sur base du contenu du descripteur : si l'information concernant le descripteur PERE est nulle, le descripteur courant occupe deux mots standard IDML; sinon il en occupe trois.

- *** procédure MAJ-ADR-MOT ***

  Cette procédure fournit dans la partie ADR-MOT du descripteur l'adresse en mémoire de la valeur qu'elle décrit.

  Il s'agit d'un descripteur qui se trouve dans une arborescence de descripteurs. Il faut remonter cet arbre jusqu'à trouver une adresse de référence, puis redescendre en faisant les mises-à-jour nécessaires.

- Plusieurs procédures ont été réalisées afin de traiter les communications avec la mémoire :

  - PUT-MEM (source, adresse-destination, format)
  - GET-MEM (destination, adresse-source, format)
  - GET-PUT-MEM (adresse-source, adresse-dest, format)

- *** procédure INTERPRETE ***

  - initialisations;

- traitement successif des instructions sur base de la valeur du PC .

  Sur base du code-processeur contenu dans l'instruction

  - TRAIT-TRANSFERT ;
  - TRAIT-ARITHMETIQUE ;
  - TRAIT-BRANCHEMENT ;
  - TRAIT-STRUCTURE ;
  - TRAIT-PRIM-BD .

  sinon une erreur est signalée et on passe au mot standard suivant.

  Ce traitement se poursuit jusque quand l'indicateur FIN-INTERPR est positionné.

- *** procédure TRAIT-TRANSFERT ***

  Sur base du code de la famille d'instruction contenu dans l'instruction courante :

  - appel FAM-MOVNIT ; PC-SUIVANT (1)
  - appel FAM-MOVI ; PC-SUIVANT (2)
  - appel FAM-MOVE ; PC-SUIVANT (2)
  - appel FAM-MOVA ; PC-SUIVANT (2)

  sinon une erreur est signalée et PC-SUIVANT (1)

  Les quatres appels ci-dessus s'adressent au processeur de transfert PROC.1.

- *** procédure TRAIT-ARITHMETIQUE ***

  Sur base du code de la famille d'instruction contenu dans l'instruction courante :

  - appel FAM-AR-ELEM ; PC-SUIVANT (1)
  - appel FAM-AR-IMM ; PC-SUIVANT (2)
  - appel FAM-AR-GEN ; PC-SUIVANT (2)

  sinon une erreur est signalée et PC-SUIVANT (1)

  Les trois appels ci-dessus s'adressent au processeur de transfert PROC.2.

- *** procédure TRAIT-BRANCHEMENT ***

  Sur base du code de la famille d'instruction contenu dans l'instruction courante :

  - appel F-BCE-RET ; la valeur de PC est
  - appel F-BCI-RET
  - appel F-BCI-VAR ; fonction du résultat
  - appel F-BC-VAR
  - appel F-BI

- positionnement de l'indicateur FIN-INTERPR

sinon une erreur est signalée et PC-SUIVANT (1)

Les cinq appels ci-dessus s'adressent au processeur de transfert PROC.3.

- *** procédure TRAIT-STRUCTURE ***

Sur base du code de la famille d'instruction contenu dans l'instruction courante :

- appel ADR-E-REP ; PC-SUIVANT (2)
- appel ADR-E-COMP ; PC-SUIVANT (1)

sinon une erreur est signalée et PC-SUIVANT (1)

Les deux appels ci-dessus s'adressent au processeur de transfert PROC.5.

- *** procédure TRAIT-PRIM-BD ***

- préparation des paramètres et appel du processeur Base de Données PROCBD (procédure CALL-PROCBD)

- PC-SUIVANT (3)

9. LE PROCESSEUR BASE DE DONNEES : PROCBD.

9.1. Fonction.

Le processeur de Base de Données supervise l'ensemble des Bases de Données du système. Toute demande d'exécution d'une primitive sur un de ces bases doit automatiquement passer par le PROCBD.

Son rôle est sutout important lors de la demande d'ouverture de la Base de Données. Le processeur Base de Données attribue à chaque interface une référence. C'est dorénavant par cette référence que sera sélectionnée l'interface à laquelle s'adresse la requête.

9.2. Informations.

- AD-ARG1, AD-ARG2, AD-ARG3, AD-ARG4, AD-ARG5 :

  adresses des cinq paramètres qui constituent la primitive à transmettre à l'interface.

- Z-CODES :

  zone de paramètre Z-CODES dont l'adresse physique d'implantation vaut AD-ARG1.

  Le processeur Base de Données connaît la structure de cette zone car il a besoin d'accéder à l'argument SREF (référence BD) et RETCODE (code retour).

9.3. La procédure principale.

- *** procédure PROCBD ***

  Cette procédure peut opérer dans deux situations différentes

  - SREF vaut 0 : le processeur interroge suuccessivement toutes les bases de Données disponibles jusqu'à ce qu'une d'elles se soit reconnue (RETCODE <> 4) ou que toutes les Bases de Données aient été sondées.

  - SREF est différent de 0 : dans ce cas, si la référence correspond effectivement à une Base de Données du système, le processeur lui transmet la requête sinon un code d'erreur est retourné à l'utilisateur.

Actuellement, l'attribution de la référence aux différentes Bases de Données est faite de façon statique :

- Base de Données Terminal : 1
- Base de Données Fichiers Standard : 2
- Base de Données conventionnelle BDTEST : 3

10. LES PROCESSEURS SPECIALISES : PROC.1, PROC.2, PROC.3, PROC.5

10.1. Fonction

Ces processeurs spécialisés prennent en charge l'exécution d'opérations bien définies et et bien délimitées.

Les possibilités d'action sont fonction du langage dans lequel est programmé le système. C'est ainsi que les formats des valeurs pouvant être traitées sont spécifiques au site.

Nous avons vu précédemment que, par exemple, l'entier occupe 16 bits alors que dans d'autres systèmes il peut en occuper 32.

Ces contraintes et limites sont prises en charge le plus possible par les processeurs afin que le système total ne soit pas interrompu à la suite d'une incompatibilité.

C'est ainsi que par exemple, la gestion des overflow lors d'opérations arithmétiques est opérée par le programme et les dépassements de la mémoire de la machine virtuelle sont surveillés par le système.

10.2. Informations et procédures principales.

Ces deux points ne sont pas abordés ici.

La raison qui nous a poussé à une telle décision est double.

Tout d'abord les principes qui ont dirigé cette réalisation ont été abordés lors de la présentation des différents processeurs (partie principale - Chapitre 2.3)

Ensuite, parce que les procédures réalisées ne mettent pas en jeu toutes les possibilités du langage de base.

C'est pourquoi, la documentation qui accompagne les programmes a été jugée suffisante pour une approche plus pratique; ce qui évite de devoir présenter ici le détail fastidieux de la réalisation.

Enfin, il est clair que ,une fois le système opérationnel, un rapport d'utilisation du langage de base précis est indispensable afin d'éviter à l'utilisateur des opérations impossibles ou dangereuses.